AFRL-RI-RS-TR-2011-226

A DISTRIBUTED MIDDLEWARE-BASED ARCHITECTURE FOR FAULT-TOLERANT COMPUTING OVER DISTRIBUTED REPOSITORIES

UNIVERSITY OF TEXAS AT ARLINGTON

SEPTEMBER 2011

FINAL TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ AIR FORCE MATERIEL COMMAND ■UNITED STATES AIR FORCE ■ ROME, NY 13441

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* SEP 2011	**2. REPORT TYPE** Final Technical Report	**3. DATES COVERED** *(From - To)* JUL 2009 – JAN 2011

4. TITLE AND SUBTITLE A DISTRIBUTED MIDDLEWARE-BASED ARCHITECTURE FOR FAULT-TOLERANT COMPUTING OVER DISTRIBUTED REPOSITORIES	**5a. CONTRACT NUMBER** FA8750-09-2-0199
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 65502F
6. AUTHOR(S) Sharma Chakravarthy	**5d. PROJECT NUMBER** 558J
	5e. TASK NUMBER 09
	5f. WORK UNIT NUMBER 02
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Texas at Arlington, 300 Nedderman Hall, 416 Yates Street, Arlington, TX 76019	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RISD, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2011-226

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project has established the foundations for developing a 'Distributed Middleware-based Architecture for Fault-tolerant Computing Over Distributed Repositories'. During this project, the sub problems and related issues were identified and the creation of an integrated solution was investigated. In this project, the contractor implemented proof-principle systems to show the effectiveness of the proposed approaches. The simulator has been extended to incorporate features specific to this scenario and analysis has been performed. This effort is by no means complete. Furthermore, not all the problems in the proposed architecture have been addressed. There are a number of important problems that need to be addressed to obtain a complete solution. In each section of the final report, the contractor has articulated the need for extending proposed solutions to reach a practically useful approach/solution.

15. SUBJECT TERMS
Fault-tolerant, distributed information management, managed information objects, scalable fault tolerance, data repositories

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** ROBERT M. FLO
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	SAR	93	**19b. TELEPHONE NUMBER (*Include area code*)** (315) 330-2334

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Summary

This project has established the foundations for developing a 'Distributed Middleware-based Architecture for Fault-tolerant Computing Over Distributed Repositories'. During the project, we have identified the sub problems and related issues, and investigated creation of an integrated solution. This effort is by no means complete. In each section, we have articulated the need for extending proposed solutions to reach a practically useful approach/solution.

Furthermore, we have not addressed all the problems identified in the proposed architecture. There are a number of important problems that need to be addressed to obtain a complete solution. We will be happy to explore opportunities with AFRL or other agencies to continue this work.

In this project, we have implemented proof-principle systems to show the effectiveness of the proposed approaches. The simulator has been extended to incorporate features specific to this scenario and analysis has been performed. The next logical step is to create a prototype by bringing all the components together and then move towards a testbed. The team has been chosen from the outset keeping the long term goals in mind and team is well-positioned for additional work on this project.

1. INTRODUCTION

The overall architecture includes a middleware in each node that has a number of services (based on SOA) for Collecting, Managing, Replicating data and Meta data for the purposes of routing and query processing. Briefly, the following services are described in this report:

- Use case scenarios
- Meta data maintained at each node
- Query plan generation algorithms, heuristics, and query execution
- Replica determination and management
- Incorporating the above into the simulator
- Software developed

First-effort solutions are described in this report along with a use case that describes how the solutions fit within an operational scenario. They can be improved further by using additional characteristics of the problem under consideration. For example, currently, a **static plan** is generated for each query and the query is processed sequentially by performing a sequence of operations at different nodes. This can be further improved by generating a **dynamic plan** at each node where the query is processed to accommodate intermittent connectivity among nodes. Furthermore, a query plan can be **executed in parallel** (using either static or dynamic plans) to improve response time.

Currently, replication is assumed to be **single copy and complete** for each relation. This can be extended to **multi-copy and partial** replica again to accommodate intermittent connectivity. Above-mentioned extensions will add complexity to both Meta data management and query processing.

Figure 1 shows the proposed Service Oriented Architecture (SOA) for the middleware for supporting query processing over distributed repositories and accommodate fault tolerance.

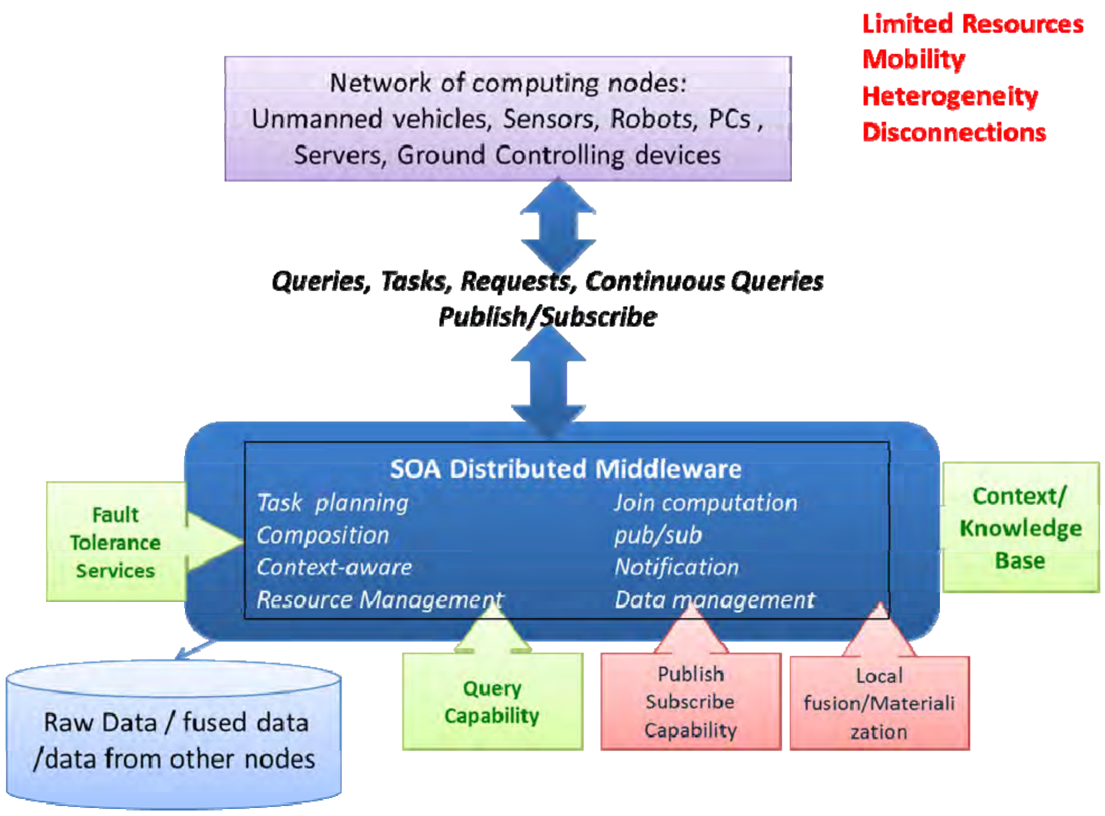

Limited Resources
Mobility
Heterogeneity
Disconnections
Network of computing nodes: Unmanned vehicles, Sensors, Robots, PCs, Servers, Ground Controlling devices
Queries, Tasks, Requests, Continuous Queries Publish/Subscribe
SOA Distributed Middleware
Task planning
Composition
Context-aware
Resource Management
Join computation
pub/sub
Notification
Data management
Fault Tolerance Services
Context/ Knowledge Base
Raw Data / fused data /data from other nodes
Query Capability
Publish Subscribe Capability
Local fusion/Materialization

Figure 1: Architecture

Message format and the algorithms presented in the previous report are assumed.

2. METHODS

The project mainly employed *analysis and simulation* during the investigations. Extensions of the project would involve studies on test-bed and eventually deployment in real world environments.

Management of metadata for the purpose of query management under volatile network conditions is discussed in Section 5. Exhaustive investigations into query plan generation are discussed in Section 6. Network management is one of the key challenges to fault-tolerant computing in a network of UAVs. We discuss methods to maintain network configurations in Section 7. Section 7 also deals replication management to ensure availability of data when nodes (UAVs) move in and out of the network. Finally in Section 8, we discuss simulation methodology and present results.

3. ASSUMPTIONS AND PROCEDURES

In order to ground the analysis for scalable fault-tolerant repositories we needed to define a use case which can help understand the requirements, issues and constraints over which we will need to operate.

Our first step was to cite some high level requirements for any such use case analysis. The use case should depict operations having:

- 10 -12 Airborne platforms (UAVs, Helos, Fighters, AWACs, …)
  Which travelling at various speeds (~100 kts, ~200 kts, ~500 kts)
  some in formation, some on independent tracks
- Ground platforms (stationary and moving)
  Stationary – semi permanent positions
  Mobile (vehicle, foot)
- Intermittent connectivity

These operations involve:

- Aggregation of distributed information
  Static & dynamic Query Plan generation
  Join queries

- Replication and information movement services

Given these requirements our approach
  - Identified actors
  - Specified information needs in scenario
  - Used representative locations for information sources
  - Shows information flow

Example scenarios that we considered were:

  - Cooperative Combat Air Patrol. *Mixture of UAVs, manned fighters, and AWACs cooperatively defend Taiwan Strait.*
    Information needed: Signals, Lines of Bearing, contact positions, tracks
  - Armed Reconnaissance / Combat Search & Rescue. *Group of UAVs search for specific targets in a large area.*
    Information needed: Detection positions, object characteristics
  - Unplanned Tactical Intelligence Support. *Ground unit requests images from any platform transiting area.*
    Information needed: Area of Interest, object positions, object characteristics, collection period, images
  - Persistent Surveillance. *Multiple UAVs conduct constant surveillance of Waziristan area over many months.*
    Information needed: Object characteristics, object positions, patterns, collection periods

- Cooperative Close Air Support. *Multiple platforms execute a coordinated attack on forces poised to overrun friendly forces on the Korean peninsula.*
  Information needed: Blue positions, red positions, object characteristics, risk levels, confidence levels
- Cooperative Air Interdiction. *Multiple platforms execute a coordinated attack on artillery targets north of the Korean DMZ.*
  Information needed: Positions, Lines of Bearing, SAM detections
- Cooperative Suppression of Enemy Air Defenses (SEAD). *Multiple platforms execute a coordinated attack on the Iranian IADS.*
  Information needed: Signals, Lines of Bearing, positions

At the August 2010 quarterly review meeting, AFRL representatives said the Cooperative Close Air Support scenario was of highest interest. We have begun work on the Cooperative Close Air Support use case and have identified the following queries (specified in operational language):

1. Get all images taken within last two days of the area bounded by latitude, longitude1 and latitude2, longitude2.
2. Get all images taken six to eight months ago of the area bounded by latitude1, longitude1 and latitude2, longitude2.
3. Get all SAM locations within 12 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.
4. Get all AAA locations within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.
5. Get the latitude, longitude, and elevation for any object of type rocket launcher in the area bounded by latitude1, longitude1 and latitude2, longitude2.
6. Get the current ceiling and visibility in the area bounded by latitude1, longitude1 and latitude2, longitude2.
7. Get all forward blue force positions within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.

The selected scenario in fleshed out in Section 4.

4. USE CASE SCENARIOS

Background

The contracted work involved research on ways of storing and retrieving information from multiple airborne and ground based networked resources. This use case provides an operational context for the technical capabilities formulated during this effort.

As seen in Figure 2, the chosen scenario involves an outbreak of hostilities on the Korean peninsula which poses different, and in some ways more difficult, challenges for the envisioned technology than those that would be experienced in current operations in Afghanistan. In a Korean conflict, we will not have air supremacy and the air defenses will be significant, making long duration overflights or orbits over enemy territory impossible.

The use case also incorporates some capabilities that don't currently exist but are anticipated to exist within the next five years. These include cooperative autonomous unmanned aerial vehicles (AUAV), improved communications and networking equipment, and technologies for fusion, distribution, and retrieval of information

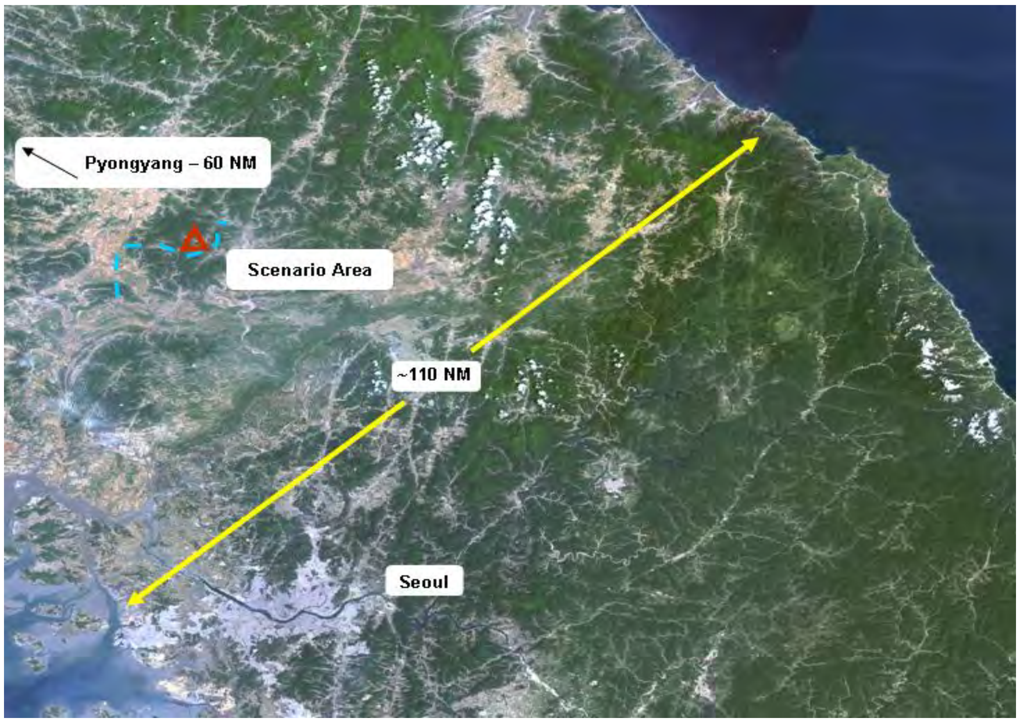


Figure 2: Scenario Location

Scenario

It's wintertime in Korea, the ground is covered in snow, and there are frequent low ceilings and visibility. An army maneuver unit advancing towards Pyongyang has been stopped in position and is taking heavy fire and casualties from dug in, concealed enemy forces.

Two Apache helicopters were lost to heavy enemy fire when they tried to take out the enemy position. The army maneuver commander requests immediate close air support to eliminate the enemy threat. The enemy position of concern is in trees with friendly troops within 100 yards. Even at this close range, friendly ground forces cannot see the firing position because of the foliage and weather conditions. The area is within range of numerous enemy anti-aircraft artillery (AAA) and surface to air missile (SAM) sites.

Two A-10Cs are tasked with providing the close air support. Two autonomous mini-unmanned aerial vehicles flying in an orbit in the low UAV corridor five miles from the Forward Line Own Troops (FLOT) will also be involved. The mission will be directed by a Joint Terminal Attack Controller (JTAC) equipped with an advanced terminal.

In preparation for the attack, the two AUAVs are tasked to conduct surveillance over the target area. They construct a plan to overfly the area from opposite directions to capture imagery from different angles, and at different times to increase their survivability.

The fused imagery is relayed to the JTAC. The JTAC also requests any imagery of the area captured prior to wintertime. The combined imagery reveals the enemy position with enough certainty to derive coordinates.

As the A-10s are proceeding to the contact point, the JTAC forwards the best image to the A-10s. The A-10s decide on a staggered attack from two different initial points (IPs). They receive the nine line information from the JTAC. The first aircraft proceeds inbound and is cleared by the JTAC after the aircraft is visually acquired. The lead takes heavy fire but is able to take a good shot. He verbally relays target area visual information to the second aircraft that is now inbound and has also been cleared upon visual by the JTAC. The second aircraft uses the fire and smoke from the first attack to deliver a devastating blow on the position.

Figure 3 illustrates the mission execution elements.

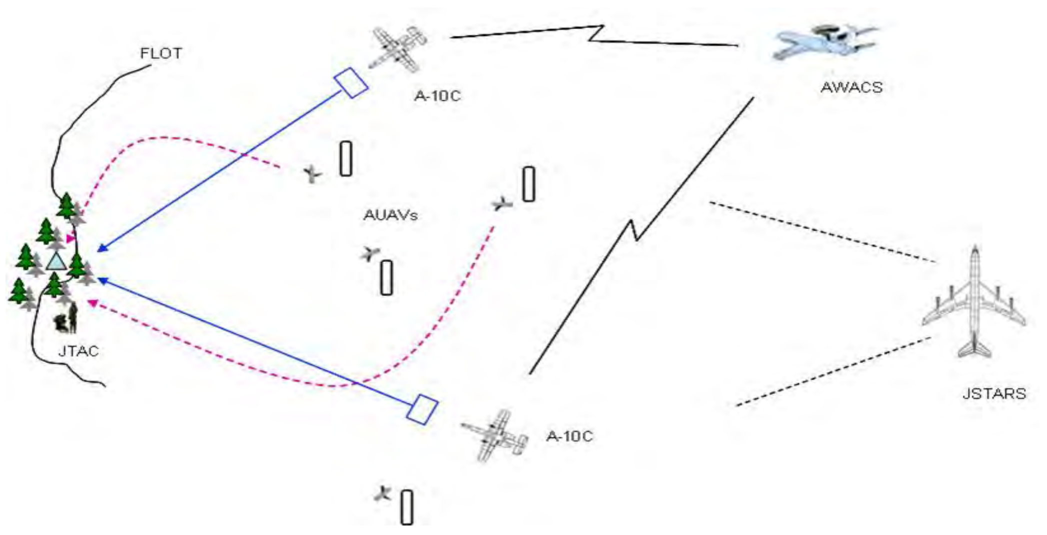
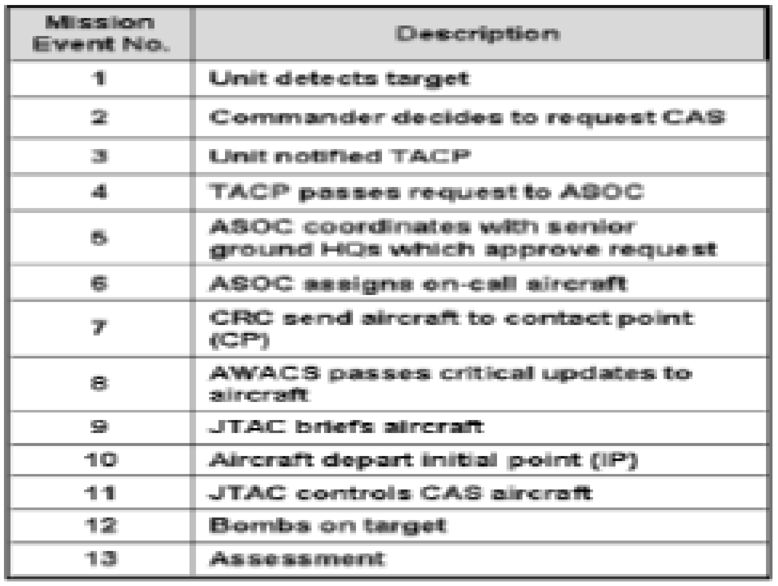


Figure 3: Mission Execution Elements and Plan

Sequence of Events

Table 1 shows the high level mission events that take place during the mission.

Table 1 Joint Mission Thread



Mission Event No.	Description
1	Unit detects target
2	Commander decides to request CAS
3	Unit notified TACP
4	TACP passes request to ASOC
5	ASOC coordinates with senior ground HQs which approve request
6	ASOC assigns on-call aircraft
7	CRC send aircraft to contact point (CP)
8	AWACS passes critical updates to aircraft
9	JTAC briefs aircraft
10	Aircraft depart initial point (IP)
11	JTAC controls CAS aircraft
12	Bombs on target
13	Assessment

A more detailed description related to the scenario is the following:

1. Army maneuver commander requests immediate close air support on a heavy weapons target located at approximate latitude and longitude coordinates.
2. The Joint Air Operations Center (JAOC) tasks two A-10Cs with the mission.

3. The requesting commander is informed about the mission and planned ordinance.
4. Mission details are passed to the JTAC and the assigned aircraft.
5. AUAVs are used under low ceilings in a high threat environment to get close up images of the target area.
6. The JTAC uses his advanced terminal to gather and collate information to precisely fix the target coordinates. The advanced terminal displays a risk envelope around the target based on planned weapons and attack parameters.
7. The attacking aircraft are given updated mission information via AWACS while enroute to the contact point.
8. The attacking aircraft and the JTAC establish communications and exchange information via voice and ground terminal-to-aircraft system links.
9. The attacking aircraft execute a coordinated attack on the target and are visually cleared to expend ordinance by the JTAC.
10. AUAVs are tasked to conduct post mission battle damage assessment.

Distributed Queries

The following queries are executed on networked, distributed battlespace information sources on behalf of the primary mission elements tasked with the mission.

1. Get all images taken within last two days of the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the JTAC to precisely fix the target.*

2. Get all images taken six to eight months ago of the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information about the target area without snow cover is needed by the JTAC to precisely fix the target*.

3. Get all SAM locations and types within 12 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the attacking aircraft.*

4. Get all AAA locations and types within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the attacking aircraft.*

5. Get the latitude, longitude, and elevation for any object of type X in the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the JTAC and the attacking aircraft to precisely fix the target.*

6. Get the current ceiling and visibility in the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the JTAC and the attacking aircraft for attack planning (weapons and tactics).*

7. Get all forward blue force positions within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2. *This information is needed by the JTAC and the attacking aircraft to avoid fratricide.*

8. Get risk envelope for planned munitions under planned delivery conditions. *This information is needed by the JTAC and the attacking aircraft to avoid fratricide.*

Discussion

The central actor for most of the distributed queries in this scenario is the JTAC as this person has ‘on-scene’ awareness of the ground situation and is best able the assess what additional information is needed to execute a successful attack. Secondary actors are the pilots of the attacking aircraft who need the best available information on the locations of air defense systems and the current weather in the target area, particularly ceiling and visibility.

The central problem that is driving the target-related queries is the need to determine the precise coordinates of a target that is concealed and in close proximity to friendly forces. To solve this problem, light-weight, expendable AUAVs are tasked to overfly the target area to collect imagery, and queries are made to distributed resources for other recent imagery of the same area, and imagery from a time period before there was snow cover. The distributed sources for this information could be airborne platforms, theater ground systems such as the Distributed Common Ground System (DCGS), and national systems.

The aggregation and fusing of retrieved imagery to derive target coordinates would normally be done in theater Intelligence, Surveillance, and Reconnaissance (ISR) and Command and Control (C2) systems. However, in thinking about future technical possibilities, and the need for increased responsiveness in an environment such as a future Korean conflict, these operations could be accomplished by the JTAC using an intelligent terminal which would also be used to relay the processed information (annotated maps, images and coordinates) to the attacking aircraft. Additionally, the attacking aircraft have automated systems that display threat and friendly position information received over data link.

5. METADATA MANAGEMENT

This section describes the data as well as the Meta data maintained at each node for the purpose of query processing.

The meta-data is maintained using a relational database. The schema for each piece of meta-data is described below.

Data at each node is assumed to be a relation with the following format (approximately 100 bytes per tuple/row). Refer to Table 2 for details.

R_{ij} correspond to relation R_i at node j. R_{ii} (i = j) will be used to represent the primary copy of a relation at node i. R_{ij} (i <> j) will be used to indicate the replica of Ri in node j.

A field 'TimeOfUpdate' is maintained for each update that happens over the Meta Data to estimate the accuracy of data and keep a track of how recent the update has been done.

5.1 Data stored at each node

5.1.1 *Original Data*

Relation R_{ii} is shown below:

Table 2 Relation Format

Timestamp	**Nodeid**	**Lat**	**Long**	**Obj_type**	**Obj_desc**	**Object_ptr**
8 bytes	4 bytes	4 bytes	4 bytes	8 chars	Varchar (64)	Pointer (8 bytes)

A number of additional information about the characteristics of each R_{ii} is maintained in a node *i* (and periodically propagated to all other nodes) for the purpose of query plan generation. Figure 2.2, shows an example of the original data at each node.

5.1.2 *Replicated Data*

If a relation R_i is replicated at this node(j), then for each replicated relation R_{ij}, we need to maintain the same information as in the previous table. The difference is that this information may not be current. Every Node maintains a copy of its original relation that is stored at some other node. R_{ij} will be used to indicate the replica of R_i in node j.

Currently, replication is assumed to be a single copy and complete for each relation.

5.2 Network Managed Data

Network Managed data is maintained and updated by the middleware and thereby accessed for processing by intermediate steps of the query plan generator.

5.2.1 Relation Attribute Information at each node

A number of additional information about the characteristics of each R_{ii} is maintained in node i and is periodically propagated to all other nodes for the purpose of query plan generation and cost estimation. Selectivity for simple and composite conditions can be inferred from Table 3.

Table 3 Original Data at each node

Attr Name	Type	Cardinality	Position	Width	Min Value	Max Value	Unique values in the range
Timestamp	number	1200	1	100	50	140	90
Lat	number	1200	2	4	10	100	90
ObjType	varchar	4000	3	64	20	350	330
ObjPtr	categorical	2000	3	8	Null	Null	10

5.2.2 Relation Information

In addition, we may maintain some information about cardinality and tuple width of each relation in that node as in table 4.

Table 4 Relation Information

TimeOfUpdate	RelationName	Cardinality	Width
	R_1	8000	70

5.2.3 Relation to Node Mapping Matrix

A Relation to Node mapping table as in the Table 5, is maintained by the message management system at each node which tells about the location of the original and the replica of a Relation. A value of 0 in the replica node column indicates that the replica is not complete at this point in time and hence is not considered for generating a query plan.

Table 5 Information about the location of replica of each node

Name	Original Node	Replica Node
R_1	1	4
R_2	2	1
R_3	3	4
R_4	**4**	**0**
….	…	…
R_n	N	k

5.2.4 Connectivity Map

A connectivity Map is maintained at each node which checks for whether there is connection between any two nodes and the current bandwidth between the two. If the Received Signal Strength (RSS) is zero or below the considerable threshold, connection is considered to be 0 and, 1 otherwise. RSS value lies on a scale of 1 to 10. LSF (the Link stability Factor) is a function of rate of change of RSS value over a period of time. A pair is considered for the plan generation if the RSS value at the instant is 1. A sample connectivity map is shown in Table 6.

Table 6 Connectivity Map

$Node_i$	$Node_j$	RSS	LSF	Bandwidth	Startup Cost
R_1	R_3	1	5	100	10

6. QUERY PLAN GENERATION, HEURISTICS, AND EXECUTION

A query plan for this scenario is envisioned as numbered sequence of plan steps that can be easily interpreted and executed at any node. Table 7 gives a description of a plan format. Each step includes the operation to be applied, the data items involved, the node where it is applied, the name of the result and the node where it is created.

6.1 Query Plan Format

Unlike traditional query processing, the plan needs to be sent from node to node (or partial plans generated at each node which is not considered in this project) for the purposes of query processing. A counter indicates the next step to be executed[1]. An example of a query plan is shown in Table 8.

Plan counter (initially set to 1 and incremented after the execution of each operation/row)

Table 7: Plan Format

Operation 1	Param	Operand1	Operand1 Loc	Operand2	Operand 2 Location	Result Name	Result Loc
Operation 2	Param	Operand1	Operand1 Loc	Operand1	Operand2 Loc	Result Name	Result Loc
…	…	…	…	…	…	…	…
Operation n	Param	Operand1	Operand1 Loc	Operand1	Operand2 Loc	Result Name	Result Loc

Table 8: Plan Format Example

Operation	Param	Operand 1	Operand 1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	$A > 100$	R_1	1	Null	Null	R_1'	1
Project	A_1, A_3, A_4	R_1'	1	Null	Null	R_1''	1
Move	Null	R_1''	1	Null	Null	R’’	2
SemiJoin	A = C	R”	2	R_2	2	SR_1	2
Join	B = D	R_{12}	2	R_2''	2	JR_1	2

The plan format described above is sufficient to describe any arbitrary relational query plan involving selects, projects, and joins (also known as an SPJ query). The above format can also accommodate SQL aggregate operators, such as a SUM, COUNT, AVERAGE, MINIMUM, and MAXIMUM needed for the next phase.

[1] It is also possible to send the entire plan (or even portions relevant to each node) to all nodes that execute portions of the plan in which case it need not be sent from node to node!

A query is executed as follows. A complete plan is generated at the node where the query is received using the metadata stored in that node. The plan is then sent to the node in which the first operation takes place (if it different from the node where the query plan is generated) along with a value indicating the starting plan step. The interpreter in that node uses the plan counter to execute as many steps as possible in that node. When a move or copy is encountered, it sends the data as well as the plan (most likely the remaining portion of the plan to reduce the amount of data transferred) to the next node. This process continues until the last step of the plan is executed. The result of the query will be in the node that executed the last step.

Currently, a complete query plan is generated as follows. Each node in the architecture has the same query plan generator and **uses the only the Metadata in that node**. Note that the metadata is updated by the underlying mechanism described in Section 2. The query plan is constructed one join/semi-join at a time. Cost computations of partial plans are done using the statistics and formulae for computing selectivity (described below). The lowest total cost query plan is used as the final plan after all possible plans are explored. This will result in an optimal plan. Several heuristics are explored as part of this project to reduce the total computation required for generating a plan. These are also compared.

The complexity of the optimal plan generation is k^n where n is the number of joins and k is the number of alternatives for each join. Currently, k being used is about 18 (multiple join alternatives, multiple semi-join alternatives and the same using replica as well) and we are assuming no more than 3 joins in a query. With this assumption, we will explore 5000+ alternative query plans and cost computation for each one of them. We have incorporated some heuristics to limit the number of plans carried forward after each join. Simulations will be performed to validate the heuristics to make sure they are meaningful.

Statistics in the form of cardinality and domain characteristics are used for cost estimation. Join and condition selectivity are inferred from the statistics maintained. Result sizes are also estimated and its accuracy is important as choice of the best query plan is primarily based on the cost of data transfer based on availability of connectivity.

Below, we outline the statistics used for evaluating the cost of a (partial) query plan. Most of these are well-established for the relational model. We do not include the processing cost for the operation/plan, but only the data transfer cost. Processing cost depends upon the availability of index and other structures and mainly influences the order of join (which we take into account in our plan generation process). Beyond this project, it will be useful to determine what access structures are meaningful and take the processing cost into account as well. In each step, the plan can be executed by converting it into an SQL statement if there a relational database is used for storing data in that node.

6.2 Statistics used for plan generation

Card(R): Number of tuples in relation *R*

Card(R.A): Number of distinct values of attribute *A* of *R*

C_o: Fixed cost to start up a connection

LSF_{ij}: Current Link Stability Factor between nodes *i* and *j* (in bytes/time unit); 0 if not connected

Select: *ρ* is a fraction of tuples satisfying a given select condition

Result size: card(S) = $\rho(C_1)$ * card(R) // when select(C_1) is applied to *R*

ρ (A = value) = 1/card(R.A)

ρ (A > value) = max(A)- value/ max(A) – min(A)

ρ (A < value) = value – min(A) / max(A) – min(A)

ρ (A ≠ value) = 1 - 1/card(R.A)

ρ (A = v_1 AND B= v_2) = 1/card(R.A) * 1/card(R.B) //assuming independence

ρ (A = v_1 OR B= v_2) = 1/card(R.A) + 1/card(R.B)

For a select operation, width of the result is the same as that of the operand

Project: For S = project(A_1, A_2, …, A_m)[R] with attributes *A_1, A_2, …, A_n* and *$m < n$*

card(S) = Card(R), if projected attributes include a key of *R*,

= max{card(R.A_i), ….,..card(R.A_j)} otherwise

Join: T= R[A=B]S

card (T) is always ≤ card (R) * card(S)

Card(R[A=B]S)= max{card(R), card(S)} , if *A* is a key of *R* and *S* a foreign key

Else, Card(R[A=B]S) ≈ card (S.B)/ card(dom(B))

Theoretically, Join Selectivity, JS= card(result)/ card(cross product)

width(T) =width(R) + width(S) //width always refers to the width of a tuple

In case of Natural Join, since natural join also eliminates one of the join attributes, the width of join attributes is subtracted from the width of result.

Cost= B_{ij} * card(S) * width(S) if Relation *S* is sent to the node of *R.*

Cost = B_{ij} * card(R) * width (R) if Relation *R* is sent to the node of *S.*

Semi-Join: T= R<A=B]S that is, *S* is projected on *B* and sent to the node of *R* for the semi-join

ρ(selectivity of semi-join) is needed to estimate the result size

ρ ≈ card (S.B)/card(dom(B)) // card(dom(B)) denotes the number of distinct values in B's domain

Card (T) ≈ ρ * card(R)

width(T)= width(R) + width(S.B)

Cost (Move(S.B in node *i* to node *j*)) = C_o + B_{ij} * card(S.B) * width(S.B).

The cost for T = R [A=B>S can be computed similarly.

6.3 Query Plan generation Design

In this section, we present the algorithm that has been developed and implemented for this project. We start with the UML diagrams, input from which a *QueryObject* is generated which is used by the *queryPlanGenerator* to generate all plans for that query using: join methods, semi-join methods, hybrid of join and semi join methods. Single-copy replication is also taken into consideration for generating the above plans.

The following UML shows the classes used for the implementation of the query plan generator.

QueryObject
relations : ArrayList<String>
projectionAttributes: HashMap<String,ArrayList<String>>
selectConjuncts : HashMap<String,ArrayList<Conjunct>>
joinConjuncts : HashMap<HashSet<String>,ArrayList<Conjunct>>
targetNode: String
dbinteractions: dbInteractions
+QueryObject(String query)
+getProjectionAttributes(String relation): ArrayList<String>
+getRelations(): ArrayList<String>
+getTargetNode():String
+getSelectConditions(String relation): ArrayList<Condition>
+getJoinConditions(String relation1): ArrayList<Condition>
+getJoinConditions(String relation1,String relation2): ArrayList<Condition>

```
+toString() : String
-isSelect(Condition testCondition): boolean
-isJoin(Condition testCondition): Boolean
```

```
^ 1
|
| 1…*
```

```
Conjunct
-----------------------------------------
conditions: ArrayList<Condition>
-----------------------------------------
Conjunct(ArrayList<Condition>)
getConditions():ArrayList<Condition>
toString() : String
```

< 1 - - 1…*

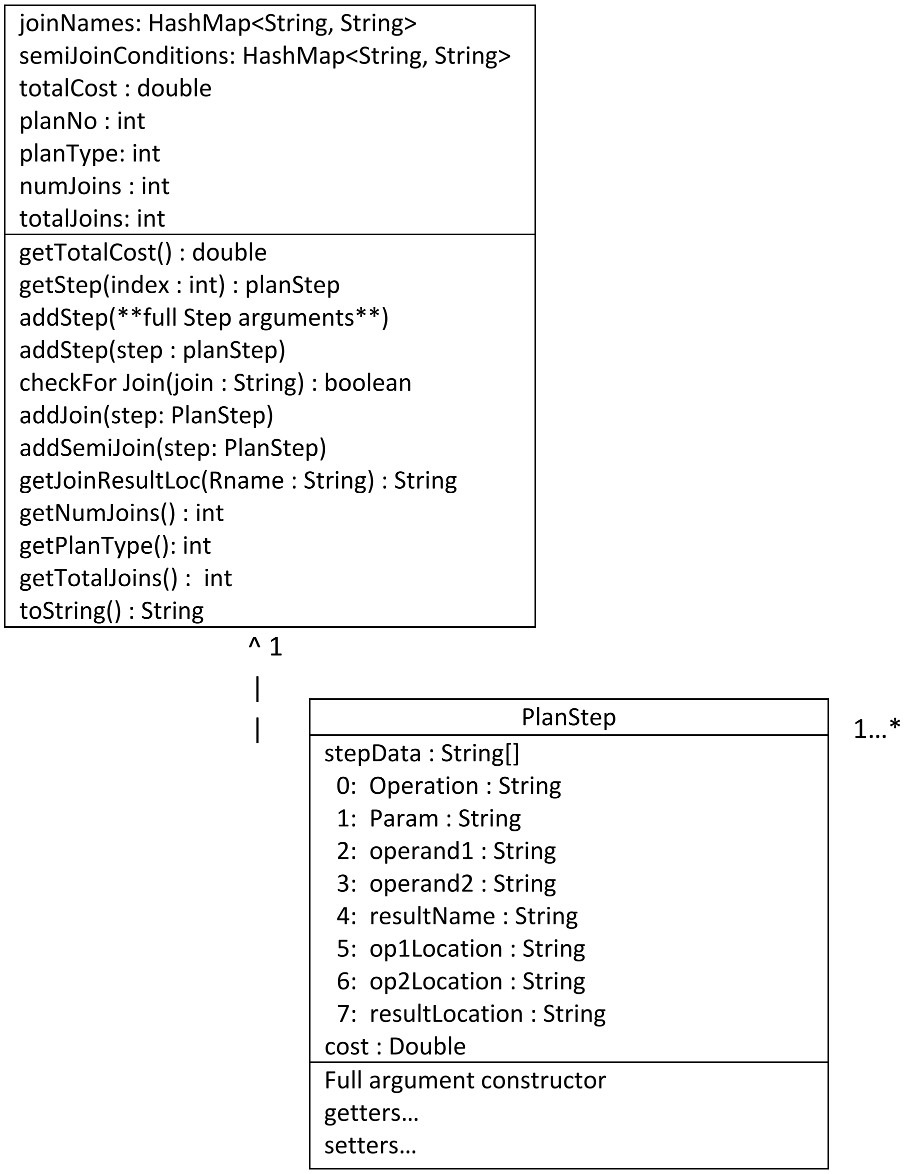
```
Condition
-----------------------------------------
leftOperand: String
rightOperand: String
operator: Operator
not: Boolean
-----------------------------------------
Condition(*Full Argument*)
getters…
toString() : String
```

```
DBInteractions
-----------------------------------------
conn: Connection
-----------------------------------------
establishDBConnection()
checkAttributeName(attributeName, relationName : String) : Boolean
getRelationName(attributeName : String) : ArrayList
relLocations(relationName : String) : ArrayList
relCardinality(relationName : String) : long
relWidth(relationName : String) : long
attributeDetails(attributeName, relationName : String) : ArrayList
connectivityDetails(node1, node2 : Int) : ArrayList
getJoinSelectivity(joinCondition : Condition) : Double
getSelectSelectivity(conjunctCondList: ArrayList<Conjunct>, relationName : String) : Double
 getCondSelectivity(selectCondition : Condition, relationName : String) : Double
```

```
PlanSpace
-----------------------------------------
planList : ArrayList<aPlan>
-----------------------------------------
PlanSpace()
createPlans(queryObject: QueryObject, dbObjetc: dbInteractions)
getNumPlans(): Int
getPlan(index : int): aPlan
sort()
toString() : String
```

```
^ 1
|
| 18^k
```

```
aPlan
-----------------------------------------
stepList : ArrayList<planStep>
semiJoins: Stack<String>
currentLocations: HashMap<String, String>
```



The query to be processed is parsed (using the JQLParser, an open-source software) and its components are packaged into a QueryObject object. The QueryObject is constructed from a String containing the SQL query - the format for the String is a standard SQL SPJ (SELECT-PROJECT-JOIN) query preceded by TARGET node number to which the query is sent and where the result is expected. Currently, the result of a query is sent to only the target node. However, it is possible to extend the algorithm to send the results to multiple nodes (and even to optimize for the set of nodes) if need be. To simplify parsing, select and join conditions are assumed to be given in conjunctive normal form (or CNF). The QueryObject constructor parses the string and populates internal data structures with the target node, projected attributes, and select/join conditions. Once the query object has been created, it is passed as an argument to the

PlanGenerator object. The PlanGenerator object can call on public methods of the QueryObject to get ArrayLists of all projection attributes, join conditions, and select conditions. Additionally, methods exist for getting all projection attributes and conditions per relation. For example, a call of getProjectionAttributes("r1") returns an ArrayList containing all projection attributes from the relation "r1".

QueryObject contains elements of a query as described below:

The FROM relations are stored as an ArrayList of Strings; no special data structures are required, and the user will simply ask for all relations.

Projection attributes are stored as a HashMap (for efficiency of lookup). Given a key (of type String) of the relation, an ArrayList of all projection attributes (type string) on that relation is returned.

Join and Select conditions each have a separate HashMap. They are stored similar to projection attributes, but with a few differences; an ArrayList of Conjuncts is returned in each case. In the case of Join, since two relations are involved in a join, the key is a pair of relations (HashSet of String), rather than a single relation.

The Conjunct and Condition classes are used to represent select and join conditions. The Conjunct class contains an ArrayList of Conditions - these Conditions are primitive conditions - all they contain are a left operand, a right operand, a comparison operator and a Boolean value representing the existence of a "NOT" operator. When an ArrayList of Conjuncts is returned by any method of the QueryObject, it is assumed that all Conjuncts contained by the ArrayList should be "AND"ed together, and that all the conditions of an individual Conjunct object should be "OR"d together.

Currently, we are parsing an input string containing SELECT, FROM, and WHERE clauses. Some reasonable assumptions are made on the select and join conditions (e.g., being n CNF). The WHERE condition is parsed by our code and the rest by the JQLParser. The next step is to use JQLParser to parse everything to populate the QueryObject.

Plan Generation Objects

PlanStep: Each PlanStep contains seven strings, one operator, parameters (as string), 2 operands, 2 operand locations, a result name and a result location. In addition each step also stores a cost. The cost is zero except for steps involving move or copy operation.

The functions for PlanStep consist of getters, setters and a toString to print, so it is purely a data storage class.

aPlan: Each aPlan contains an ArrayList parameterized to PlanStep, which is the body of a plan, a total cost, a plan number, a stack, and three HashMaps. The stack keeps track of which relations are needed for the second half of a semijoin operation. The first HashMap maps relation names as keys to the name of the relation they have been joined to or themselves if they have not been joined. The second HashMap maps relation names to their locations. The third HashMap maps leftover semijoin relations to their conditions after the first part of a semijoin has been performed since after the first semiJoin has been performed they may not have the same relation name.

The functions for aPlan consist of getters, setters and functions for adding steps which ensure that the maps are up to date. It also contains a toString function that will print all of the steps in a plan and a compareTo function that allows the plans to be sorted by their total cost.

PlanSpace: Contains an ArrayList of aPlan that stores all the plans generated during the algorithms runtime.

The functions of a Plan consist of getters, setters, a function to sort all plans by their cost and a function to populate the list of plans with the proper amount for a query. It also contains a toString function to print all of the plans.

6.4 Plan Generation Algorithm

The generator begins by generating 18 distinct partial plans (initially empty) for each join. As an exhaustive algorithm, it generates 18^k plans for a query containing k joins. It is evident that this approach is not viable beyond a few joins. This is being done so that later we can compare heuristics-based plans with the optimal ones to analyze the effectiveness of heuristics we come up with (e.g., top-k in each iteration, top-k cumulatively, top-k for each type of plan.). The generator then iterates through the relation list and creates the necessary select statements. Then all of the attributes required are projected on the output and join condition attributes to minimize the data transfer across nodes which form the bulk of the cost of query processing in this environment. Since most of the plans will use these initial select or project statements (to reduce the width and cardinality of the relation), these same statements are attached to every plan.

For plan alternatives using joins the generator moves the required relations to the location of the join and then performs the join. Even for this, projections are applied to reduce the overall width and cardinality of relations moved. The plan class takes care of updating intermediary name, location, and condition information. Then the generator moves on to the next plan.

For plan alternatives using semijoins, the relation that will be semijoined to is copied and projected on the attributes used in the specific join condition to minimize data transfer. Then it is moved to the location of the semijoin. The semioin is performed. When the semijoin step is added to a plan the plan updates name, location, and condition information and in the case of semijoins the output relation and the relation that still needs to be semijoined to finish the operation is added to a stack to keep track of the remaining semijoins (to generate chained semi join plans) . The next plan is then processed.

For multiple joins, after all of the plans have been processed with the first join, all of the joined relations will be projected on the remaining join attributes required and then algorithm will iterate through all the plans again performing the remaining joins and semiJoins. After a relation has been joined, its current location is considered to be that of the result of the join even if it currently has a replica, which may cause some of the plans to be the same.

After all cases have been exhausted, the algorithm goes through and finishes each case by iterating through the stack of remaining semijoins completing the remaining semijoins in reverse order and then moves the final relation to its output node.

During each step of the plan generation, the cost associated with a move or a copy is calculated, if there is no direct connection between nodes the cost is considered prohibitively high and value is automatically forced to a very high level by using a very low bandwidth for the calculation. After calculation the plans can be viewed in sorted form. The plan generator generates a summary of: number of plans generated, lowest and highest cost plan numbers. It is possible to view any of the plans in detail. The same process is used for generating plans using heuristics except that a subset of plans are used in each iteration which are selected based on the specific heuristic.

6.5 Examples of Generated Plans

Consider a query Q that is sent to node 1 and the results of the query will be sent to node 1.

Q:
```
TARGET 1
SELECT *
FROM U_1_D,U_2_D,U_5_D"
WHERE ((U_5_D.OBJTYPE=1))"
        AND ((U_1_D.LAT>U_2_D.LAT))"
        AND ((U_2_D.LONG>U_5_D.LONG))");
```

Below, we present three plans with their cost and some information about how they were generated in terms of plan combinations.

The lowest cost plan is

```
Plan Number 253
Total Cost 493.87333333333333 (in milli seconds)

Operator  | Parameter           |  Operand 1           | Loc1 | Operand 2            | Loc2 |  Result Name               | LocR | Cost
----------+---------------------+----------------------+------+----------------------+------+----------------------------+------+---------
SELECT    |   (U_5_D.OBJTYPE=1) |                U_5_D |    5 |                 null | null |                      U_5_D |    5 | 0.000000
COPY      |                null |             R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    5 | 0.000000
PROJECT   |                 LAT |          C->R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    5 | 0.000000
MOVE      |                null |          C->R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    2 | 393.333
SEMIJOIN  |(U_1_D.LAT>U_2_D.LAT |                U_2_D |    2 |          C->R->U_1_D |    2 |          U_2_D_SJ_R->U_1_D |    2 | 0.000000
MOVE      |                null |    U_2_D_SJ_R->U_1_D |    2 |                 null | null |          U_2_D_SJ_R->U_1_D |    5 | 50.140000
JOIN      |(U_2_D.LONG>U_5_D.LO |    U_2_D_SJ_R->U_1_D |    5 |                U_5_D |    5 |U_2_D_SJ_R->U_1_D_J_UAV_5_D|    5 | 0.000000
SEMIJOIN  |(U_1_D.LAT>U_2_D.LAT |             R->U_1_D |    5 | U_2_D_SJ_R->U_1_D_J_ |    5 |U_2_D_SJ_R->U_1_D_J_UAV_5_D|    5 | 0.000000
MOVE      |                null | U_2_D_SJ_R->U_1_D_J_ |    5 |                 null | null |U_2_D_SJ_R->U_1_D_J_UAV_5_D|    1 | 50.400000

This plan is a combination of alternatives 15 (for the first join) and 2 (for the second join)
```

This plan using only semi joins has only a slightly higher cost

```
Plan Number 263
Total Cost 585.942 (in milli seconds)

Operator  | Parameter           |  Operand 1          | Loc1 | Operand 2            | Loc2 |  Result Name               | LocR | Cost
----------+---------------------+---------------------+------+----------------------+------+----------------------------+------+---------
SELECT    |   (U_5_D.OBJTYPE=1) |               U_5_D |    5 |                 null | null |                      U_5_D |    5 | 0.000000
COPY      |                null |            R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    5 | 0.000000
PROJECT   |                 LAT |         C->R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    5 | 0.000000
MOVE      |                null |         C->R->U_1_D |    5 |                 null | null |                C->R->U_1_D |    2 | 393.333333
SEMIJOIN  | (U_1_D.LAT>U_2_D.LAT|               U_2_D |    2 |          C->R->U_1_D |    2 |          U_2_D_SJ_R->U_1_D |    2 | 0.000000
COPY      |                null |   U_2_D_SJ_R->U_1_D |    2 |                 null | null |       C->U_2_D_SJ_R->U_1_D |    2 | 0.000000
PROJECT   |                LONG | C->U_2_D_SJ_R->U_1_D|    2 |                 null | null |       C->U_2_D_SJ_R->U_1_D |    2 | 0.000000
MOVE      |                null | C->U_2_D_SJ_R->U_1_D|    2 |                 null | null |       C->U_2_D_SJ_R->U_1_D |    5 | 40.390000
SEMIJOIN  |(U_2_D.LONG>U_5_D.LO |               U_5_D |    5 | C->U_2_D_SJ_R->U_1_D |    5 |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    5 | 0.000000
MOVE      |                null |U_5_D_SJ_U_2_D_SJ_R- |    5 |                 null | null |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    2 | 60.346667
SEMIJOIN  |(U_2_D.LONG>U_5_D.LO |   U_2_D_SJ_R->U_1_D |    2 | U_5_D_SJ_U_2_D_SJ_R- |    2 |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    2 | 0.000000
MOVE      |                null |U_5_D_SJ_U_2_D_SJ_R- |    2 |                 null | null |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    5 | 41.040000
SEMIJOIN  |(U_1_D.LAT>U_2_D.LAT |            R->U_1_D |    5 | U_5_D_SJ_U_2_D_SJ_R- |    5 |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    5 | 0.000000
MOVE      |                null |U_5_D_SJ_U_2_D_SJ_R- |    2 |                 null | null |U_5_D_SJ_U_2_D_SJ_R->UAV_1_|    1 | 50.832000

This plan is generated from alternatives 15 and 11
```

The highest cost plan is

```
Plan Number 3
Total Cost 175117.8 (in milli seconds)

Operator  | Parameter          | Operand 1           | Loc1 | Operand 2            | Loc2 | Result Name              | LocR | Cost
----------+--------------------+---------------------+------+----------------------+------+--------------------------+------+---------
SELECT    |  (U_5_D.OBJTYPE=1) |               U_5_D |    5 |                 null | null |                    U_5_D |    5 | 0.000000
MOVE      |               null |               U_1_D |    1 |                 null | null |                    U_1_D |    2 | 50010.000000
JOIN      |(U_1_D.LAT>U_2_D.LAT|               U_1_D |    2 |                U_2_D |    2 |            U_1_D_J_U_2_D |    2 | 0.000000
MOVE      |               null |       U_1_D_J_U_2_D |    2 |                 null | null |            U_1_D_J_U_2_D |    1 | 57.800000
MOVE      |               null |               U_5_D |    5 |                 null | null |                    U_5_D |    1 | 125050.000000
JOIN      |(U_2_D.LONG>U_5_D.LO|       U_1_D_J_U_2_D |    1 |                U_5_D |    1 |    U_1_D_J_U_2_D_J_U_5_D |    1 | 0.000000

This plan is generated from alternatives 1 and
```

6.6 Heuristics-Based plan generation

The purpose of generating a very large plan space (in the above description) is to demonstrate the cost differences between best and worst plans. The above algorithm is still not exhaustive in that it does not consider all join combinations. As can be seen clearly, there is a significant difference between the best and the worst pan. During testing, we also realized that the connectivity plays a critical role in that if only one way connection is available between nodes, it impairs good plan generation.

We have added a number of heuristics to the above algorithm to compare their performance to the optimal plan generated by the above algorithm. We can use this implementation to analyze various aspects such as connectivity, bandwidth, as well as selectivity to understand the types of plans generated. We have identified the following heuristics to be useful and have implemented then so that we can compare them to the plan produced by the above algorithm.

Choose top k lowest cost partial plans in each round for expansion or iteration (Top-k-iteration)

Choose top k lowest **cumulative** cost (up to that round) plans in each round of expansion (Top-k-cumulative)

Categorize plans into join-based, semi-join based, and hybrid. Choose top k lowest cost plan from each category for expansion in each round (Top-k-join-type)

In addition to the above, a number of other possibilities such as incremental plan generation, looking ahead at connectivity and pruning plan alternatives, getting dynamic cost information and then generating partial plans are possible. Note that connectivity will play a significant role not only in the generation of a complete plan, but also its cost. If sufficient connectivity does not exist among the nodes that participate in the query (including the nodes that have a replica), a complete query plan may not even be feasible. The presence of replica increases the probability of generating a complete plan and if several exist, heuristics hopefully will choose a good one without having to generate all plans. A heuristic that involves connectivity would be very useful for this environment.

In Top-k-iteration, k (specified as a parameter) lowest cost plans in that round are chosen in each round for further expansion. This works as a local heuristic and hopes that the lowers cost plans will continue to show good cost value.

In Top-k-cumulative, k (again specified as a parameter) lowest cumulative cost plans up to that round are chosen in each round. This, hopefully, will improve upon the previous one in choosing plans that are good up to that round.

The Top-k-join-type is a different type of heuristic as we have different types of partial plans and their costs are likely to be different. Here, k lowest cost plans from each type is chosen for the next round. In order to compare them in a fair manner, the k value need to be lower (1/3 as we have 3 join types) so that the same number of plans are carried forward in each round.

The above three heuristics have been implemented. The software has two modes: interactive and experimental to make it easy to test and use. In the interactive mode, a query can be given at the prompt (or in a file) and a heuristic specified for its plan generation. The generator will indicate the number of plans generated as well as the lowest and highest cost plans (along with plan number). One can output (or look at) any plan in details by typing the plan number. It is also possible to provide a file input to process multiple queries in this mode. The selectivity and cardinality information is statically initialized. The connectivity is also statically initialized at the start of the system. This can be easily changed by loading a new or different relation before executing the plan generator.

In the experimental mode, one can either provide input at the prompt or submit a file for processing. The input consists of (whether given as prompted or in a file): number of queries to be generated, seed for query generation, number of connectivity configurations to be used in the experiment, seed for configuration, and connectivity factor. The generator has a random query generator on the schema stored in the system and generates the desired number of queries. The seed is to ensure repeatability of experiments as well as generate a new sequence of pseudo-random queries. The same is true for network configurations and its seed. The connectivity factor is use to control the sparseness of the connectivity matrix. If there are n nodes, the connectivity factor can vary from 0 to (n-1), 0 indicating no connectivity at all and (n-1) indicating complete connectivity. The connectivity itself is generated randomly to satisfy the parameters specified.

The above setup allows one to perform different types of experiments. For each query, connectivity can be changed to determine how the plan cost changes and can also compare the optimal cost with heuristics-based plan costs. It is possible that due to the connectivity, a number of plans cannot be completed resulting in a high cost. Queries or connectivity sequences can be changed, independently, by varying the corresponding seed.

6.7 Experimental Results

In order to test the effectiveness of the heuristics proposed for the query plan generator, we performed two sets of experiments that tested these heuristics across different connectivity matrices and several different queries.

6.7.1 *Impact of Connectivity on Plan Cost*

In this experiment, we generated a single query (shown below) and computed the top 3 plan cost using the heuristics proposed along with the optimal plan cost on six different configurations of the connectivity matrix. We had to keep the connectivity large; otherwise, no plan was generated. Since the connectivity matrix is large in size, we do not show it here. Instead, we display the configuration file used for this experiment.

```
Query 1: target 2
       SELECT * FROM UAV_2_DATA, UAV_4_DATA, UAV_6_DATA WHERE
       ((UAV_2_DATA.NODEID=76)) AND ((UAV_2_DATA.LONG>=804)) AND
       ((UAV_6_DATA.LONG<=540) AND
       ((UAV_2_DATA.LAT=UAV_4_DATA.LAT)) AND (UAV_4_DATA.LONG=UAV_6_DATA.LONG));
```

Configuration File:

```
package afrl;
public interface afrlConstants
{
       int     NUMBER_OF_QUERIES     = 1;
       //Number of queries to be generated by the random queryGenerator
       String  FILE_NAME             = "outputFiles/apr13_queries_exp1.txt";
       //Name of the file the generated queries will be stored in
       String  NETWORK_FILE_NAME     = "outputFiles/network/apr13_network_exp1";
       //Name of the outpur file the connection matrix is stored in
       int      SEED                  = 4406235;
       //Seed for query generator
       int     NETWORK_DEGREE        = 11;
       //Number of connected nodes
       int     NUM_NETWORKS          = 6;
       //number of connection matrices to be generated per query
       int     NETWORK_SEED          = 33152035;
       //seed for connection matrix generator
       int     NUM_NODES             = 13;
       //number in noes in connection matrix
       int     TopKOptimal           = 3;
       //number of optimal plans to display; 0 to show all
       int     TopKCumulativeCost    = 3;
       //Set heuristic to K value to keep or zero for inactive
       int     TopKIterationCost     = 3;
       //Set heuristic to K value to keep or zero for inactive
       int     TopKJoinType          = 9;
       //show k number of join type heuristic applying cumulative heuristic to k/3 of each type
       boolean displayNonConnective  = false;
       //Set to true to display non connective plans
       boolean heuristicDebug        = false;
       //set to true to dump heuristic execution data to files
 }
```

The results of the experiment are shown below.

Table 9 Costs incurred by various approaches

Network	Optimal Join	Optimal Semi-Join	TopK Cumulative Join	TopK Cumulative Semi-Join	TopK Iterative Join	TopK Iterative Semi-Join	TopK Join-type Join	TopK Join-type SemiJoin
1		2.81	29.01	2.81		2.81	29.01	2.81
2	7.80	1.75	10.88	268.38		1.75	10.88	1.75
3		1.47	10.88	268.38		10.88	11.73	1.47
4		1.61	13.58	1.61		1.75	13.58	1.61
5		1.01	12.32	1.01		1.01	12.32	1.01
6		1.81	10.65	268.38		1.47	10.65	2.02

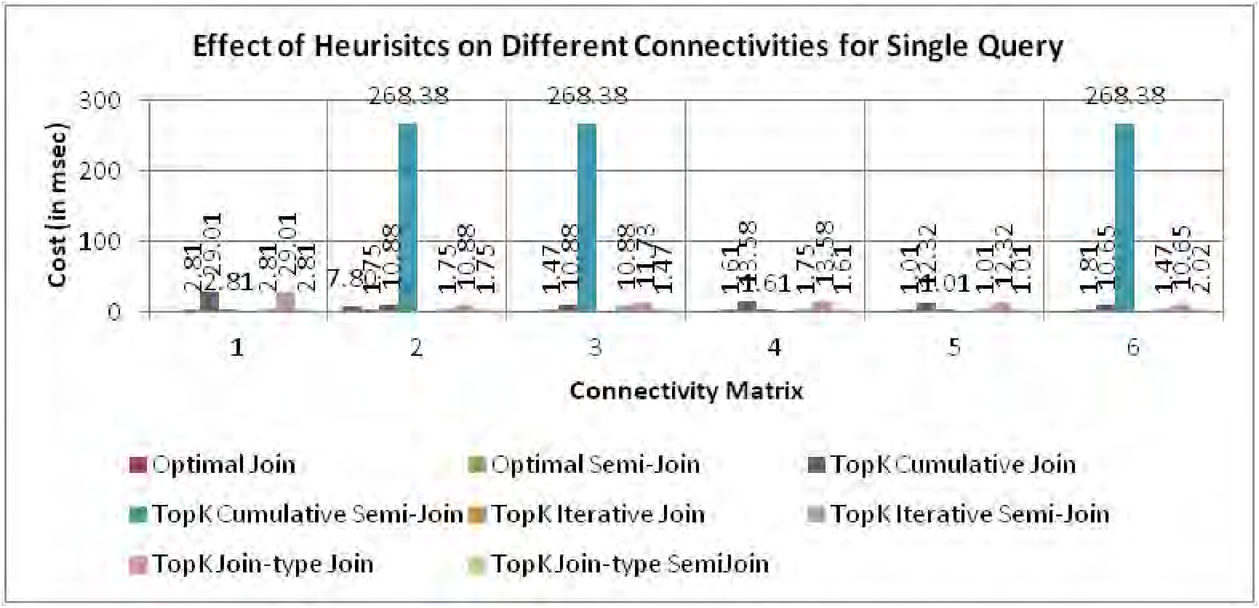


Figure 4: Effect of Heuristics on different connectivities

The graph as well as the table, shows the cost (in milliseconds) incurred by the various approaches towards generating the top-3 best plans for the given query.

Based on the results, it can be observed that the plan generation process depends heavily on the connectivity between nodes. For many network configurations, no plan is generated even in optimal join case. However, amongst the different heuristics, the Semi-join approaches appear to be suitable for the current set of connectivity configurations. However, determining the exact relationship between the type of join and the corresponding costs of plan generation will require further study.

6.7.2 *Plan cost across multiple network configurations for Different Queries*

In this experiment, we generated 5 different queries (shown below) and tested the three heuristic algorithms along with the optimal algorithm on a same configuration of the connectivity matrix. Like the previous experiment, we display the configuration file used for this experiment.

```
Query 1: target 2 SELECT * FROM UAV_2_DATA, UAV_4_DATA, UAV_5_DATA WHERE
((UAV_2_DATA.NODEID=66)) AND((UAV_2_DATA.LONG>=614)) AND ((UAV_5_DATA.NODEID=77)) AND
((UAV_2_DATA.LAT=UAV_4_DATA.LAT)) AND ((UAV_4_DATA.NODEID=UAV_5_DATA.NODEID));

Query 2: target 5 SELECT * FROM UAV_5_DATA, UAV_10_DATA WHERE ((UAV_10_DATA.LAT=609))
AND ((UAV_10_DATA.OBJPTR<=246)) AND ((UAV_5_DATA.OBJPTR=UAV_10_DATA.OBJPTR));

Query 3: target 9 SELECT * FROM UAV_9_DATA, UAV_10_DATA, UAV_5_DATA WHERE
((UAV_9_DATA.LONG>351)) AND ((UAV_9_DATA.LAT>=40)) AND ((UAV_5_DATA.LONG<=804)) AND
((UAV_9_DATA.OBJPTR=UAV_10_DATA.OBJPTR)) AND ((UAV_10_DATA.LAT= UAV_5_DATA.LAT));

Query 4: target 6 SELECT * FROM UAV_6_DATA, UAV_10_DATA, UAV_4_DATA WHERE
((UAV_6_DATA.LAT<55)) AND ((UAV_6_DATA.NODEID<=260)) AND ((UAV_4_DATA.NODEID=22)) AND
((UAV_6_DATA.TIMESTAMP=UAV_10_DATA.TIMESTAMP)) AND (UAV_10_DATA.OBJPTR=
UAV_4_DATA.OBJPTR));

Query 5: target 9 SELECT * FROM UAV_9_DATA, UAV_3_DATA, UAV_2_DATA, UAV_4_DATA WHERE
((UAV_9_DATA.TIMESTAMP<=764)) AND ((UAV_9_DATA.LONG<102)) AND ((UAV_2_DATA.NODEID=66))
AND ((UAV_2_DATA.LONG>=614)) AND ((UAV_9_DATA.LAT=UAV_3_DATA.LAT)) AND
((UAV_3_DATA.LAT=UAV_2_DATA.LAT)) AND ((UAV_2_DATA.OBJPTR=UAV_4_DATA.OBJPTR));
```

Configuration File:

```
package afrl;

public interface afrlConstants
{
  int     NUMBER_OF_QUERIES     = 10;
//Number of queries to be generated by the random queryGenerator
  String  FILE_NAME             = "outputFiles/apr13_queries_exp2.txt";
//Name of the file the generated queries will be stored in
  String  NETWORK_FILE_NAME     = "outputFiles/network/apr13_network_exp2";
//Name of the outpur file the connection matrix is stored in
  int    SEED                  = 4406235;
//seed for query generator
  int     NETWORK_DEGREE        = 11;
//Number of connected nodes
  int     NUM_NETWORKS          = 1;
//number of connection matrices to be generated per query
  int     NETWORK_SEED          = 32152035;
//seed for connection matrix generator
  int     NUM_NODES             = 13;
//number in noes in connection matrix
  int     TopKOptimal           = 3;
//number of optimal plans to display 0 to show all
  int     TopKCumulativeCost    = 3;
//Set heuristic to K value to keep or zero for inactive
  int     TopKIterationCost     = 3;
//Set heuristic to K value to keep or zero for inactive
  int     TopKJoinType          = 9;
//show k number of join type heuristic applying cumulative heuristic to k/3 of each type
  boolean displayNonConnective  = false;
//Set to true to display non connective plans
  boolean heuristicDebug        = false;
//set to true to dump heuristic execution data to files
 }
```

The results of the experiment are shown below.

Table 10 Costs incurred by various approaches for Top K Plans

Query	Optimal Join	Optimal Semi-join	TopK Cumulative Join	TopK Cumulative Semi-Join	TopK Iterative Join	TopK Iterative Semi-Join	TopK Join-type Join	TopK Join-type Semi-Join
Query 1	6.19	1.19	172.00	7.91	6.19	1.19	172.00	7.91
Query 2	52.34	6.41	4853.07	390.31		6.41	4853.07	390.31
Query 3	60.76	125.39	8557.31	785.46	60.76	125.39	60.76	125.39
Query 4	0.89	1.14	14362.78	1.21	1.34	16201.62	1.12	1.21
Query 5	0.52	0.87	0.70	1.29	0.62	1.95	0.70	1.29

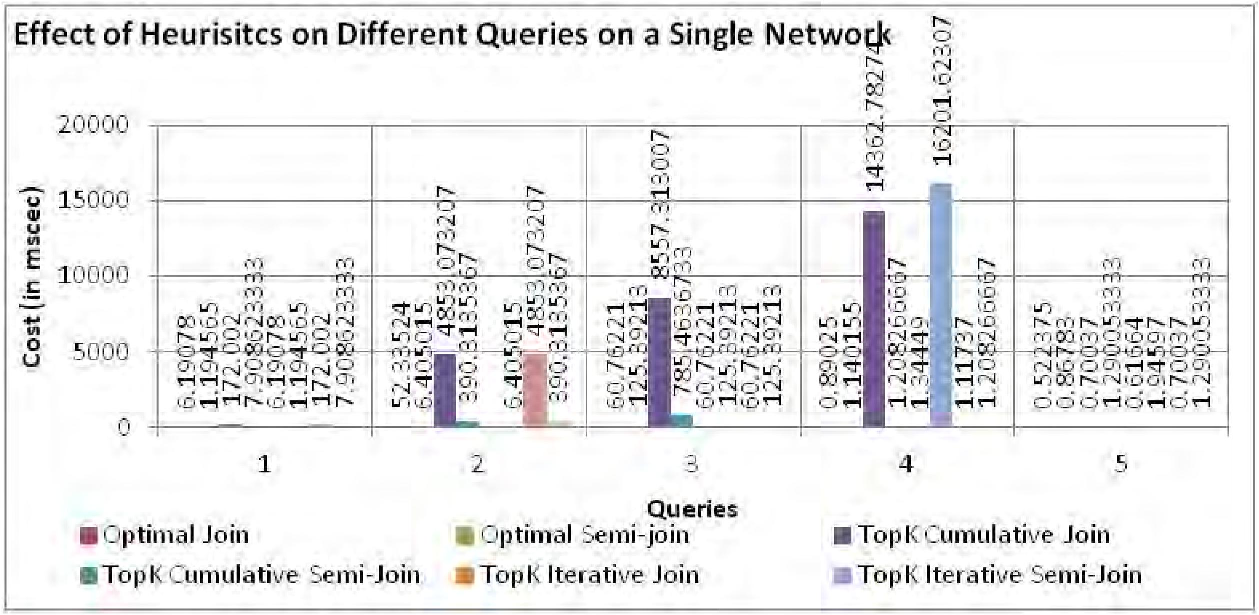


Figure 5 Effect of Heuristics on different queries

The graph as well as the table, shows the cost (in milliseconds) incurred by the various approaches towards generating the top-3 best plans (average) for 5 different queries on the same connectivity matrix configuration.

Based on the results, it can be observed that the plan generation process depends hugely on the connectivity between the nodes. Similar to the previous experiments, amongst the different heuristic approaches, the Semi-join approaches appear to be suitable for the current set of connectivity configurations.

As connectivity plays a critical role even for obtaining a plan (not necessarily a *good* plan), replication becomes even more important. We have assumed single-copy replication in this work. This needs to be extended to multiple copies. Also, it becomes important to include predicted connectivity into plan generation heuristics. Currently, it is *not* included. A different set of heuristics bases on stability of the connections, multiple replicas (both partial and complete) seem to be important. Also, parallel pan execution and dynamic plan generation (postponing the use of connectivity information until needed) seems to be critical for this scenario.

7. REPLICATION AND NETWORK MANAGEMENT

Message format for different types of messages is shown below in Figure 6. Source ID and destination ID are 5-bit unique addresses of the corresponding nodes (UAVs). Multiple flags are used to indicate the type of data in the payload.

SP	SP	D	N	I	M	A	Q	Source ID	Destination ID	Timestamp	Payload

Figure 6 : Message Format

Table 10: Interpretation of flags

Flag bit	Length	Payload Content	Description
D-Discover	1 bit	Meta data	Message to discover new nodes in the network.
N-Neighbor	1 bit	Meta data and Adjacency matrix	Message to exchange connectivity information between various nodes in the network.
I-Direction	1 bit	Node Direction	Message to exchange the direction of travel of a node.
M-Data	2 bits	Result of query execution	M=1: Message is exchanged directly with the destination node as a part of the MOVE operation. M =11: Message is exchanged directly with the destination node as a part of the COPY operation. M =10: Message is sent through a series of intermediate nodes to the destination node.
A-Alive	1 bit	No Payload	Message to indicate a node's presence to its neighbors.
Q-Quit	1 bit	No Payload	A node moving out of range sends this message to its neighboring nodes.

Setup

Initially, each node is by itself and not connected to other nodes in the network. At time T=t1 (say) when another node comes within communication range, a neighbor discovery message DISCOVER is sent by one or both the nodes. The **DISCOVER** message as shown in Figure 7comprises,

- The source identifier (ID)
- The destination ID is typically a Broadcast address

- A Timestamp
- Meta Data information
- **D** =1, indicating that the payload contains Meta data information.

SP	SP	1	N	I	M	A	Q	Source ID	Destination ID	Timestamp	Payload

Figure 7: DISCOVER

Each node receiving the **DISCOVER** message, responds with a **NEIGHBOR** message as shown in Figure 8. The NEIGHBOR message contains,

- The source identifier (ID)
- The destination ID is typically a Broadcast address
- A Timestamp
- **N** =1, indicating that the payload contains information about the node's connectivity as well as metadata information.
- **I** =1, indicating that the payload contains a node's intended direction of travel as shown in Figure 9.

SP	SP	D	1	I	M	A	Q	Source ID	Destination ID	Timestamp	Payload

Figure 8: NEIGHBOR

SP	SP	D	N	1	M	A	Q	Source ID	Destination ID	Timestamp	Payload

Figure 9: DIRECTION

A series of **DISCOVER**/**NEIGHBOR** messages among nodes within communication range results in an ad hoc network comprising connected nodes (UAVs). Similar **DISCOVER**/**NEIGHBOR** messages exchanged periodically either to gain new or retain existing connections.

Each node in the ad hoc network has information about both the connectivity as well as metadata of each of its neighboring node.

Each node maintains its connectivity with other nodes in a connectivity graph G_N, and exchanges connectivity information (The local G_N) periodically with its neighbors. It is likely that there are one or more such ad hoc networks and isolated nodes in the system at any given time. G_N is constructed through a series of **DISCOVER**/**NEIGHBOR** message exchanges among nodes. Initially, each node starts as an isolated node and adds neighbors.

```
Algorithm executed at each node after DISCOVERY

DISCOVER_NEIGHBOR

{     At node $N_i$

          if received DISCOVER from Node $N_j$

                    Update adjacency list at Node $N_i$

                    For each node $N_k$ in the adjacency list of node $N_i$

                              $N$=1

                              Send NEIGHBOR message (Updated adjacency list, Meta data
                    information)

}
```

Data exchange between nodes

Consider a snap shot of the system shown in Figure 1o. Let N_1, N_2, N_3 be three nodes in the physical network. Data items (relations) *R1* and *R2, R3* are represented as data nodes and reside on nodes N_1, N_2 and N_3 respectively.

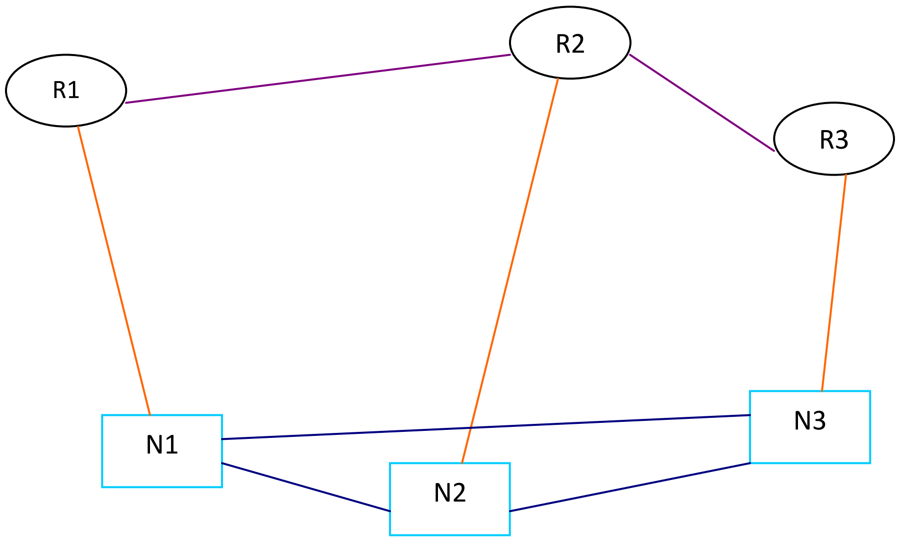


Figure 10: Network snapshot

Each physical node in the network exchanges information about what Relational data it contains as a part of meta data exchange which is included in the **DISCOVER** message. Hence node N_1 knows that nodes N_2 and N_3 contain data items R_2 and R_3 respectively. Similarly the other two nodes get to know that node N_1 contains data item D_1.

When a query arrives at node N_1, a query plan is generated based on the current connectivity of the network. The generated plan contains operations that need to be executed at various nodes with the query execution beginning at N_1. Operations local to the node N_1 are carried out first and when the first MOVE command is executed, the results of the partial query execution needs to be transferred to the next physical node in the plan.

The flag **M** is set to indicate that the payload contains the following as shown in Figure 11:

Result of the local operations at node N_1.

The query plan

A counter value to indicate that the next node in sequence needs to execute the query starting at the instruction pointed to by the counter value.

SP	**SP**	**D**	**N**	**I**	**1**	**A**	**Q**	N_1	N_2	Timestamp	Queryplan,Res_1,Counter

Figure 11: Data Message

For example:

Let the query Q that was issued to node N_1 be $Q = \pi_{A,B}(\sigma_{c1<10}(R_1)) \ltimes \pi_B(\sigma_{c2=20}(R_2))$

From the above mentioned figure, R_1 is on Node N_1 and R_2 is on node N_2.

Let the result at Node N_1 be $Res_1 = \pi_{A,B}(\sigma_{c1<10}(R_1))$

Res_1 along with the query plan and the counter needs to be sent to Node N_2 for further execution of Q.

At Node N_2 the following query is executed to give the final result of query Q.

$Res_2 = Res_1 \ltimes \pi_B(\sigma_{c2=20}(R_2))$

Algorithm to execute MOVE

MOVE (*Res_1, Node N_i, Node N_j*)

{ **Node N_i**: The node where the result of the first part of the query Q was executed.

```
Node $N_j$: The next node in the execution sequence.
    At node $N_i$
    If $E_{ij}$ exists; $E_{ij}$ is the edge between $N_i$ and $N_j$
        M=1
        Transfer $Res_1$ from $N_i$ to $N_j$
        Delete Result $Res_1$ from $N_i$
    If path $P_{ij}$ exists; $P_{ij}$ is the path from $N_i$ to $N_j$
        M=10
        Transfer $Res_1$ from $N_i$ to next node in path $P_{ij}$
        Delete Result $Res_1$ from $N_i$
    else
        MOVE cannot be executed.
At node $N_k$;can be intermediate node or destination node
If DestID=SelfID copy $Res_{1.}$
If DestID≠SelfID transfer $Res_1$ to next node in $P_{ij}$
}
```

The COPY operation executed at a node as specified by the query plan indicates that the result of query execution at that particular node needs to be sent to a set of other nodes.

The flag **M** is set to **11** and the payload contains the Result as per Figure 12.

SP	**SP**	**D**	**N**	**I**	**11**	**A**	**Q**	N_1	N_2	Timestamp	Result

Figure 12: Message sent for the COPY operation

Algorithm to execute COPY

```
COPY (*Result, NodeList[ ]*)
{   $N_i$: Node at which the copy operation is executed.
    NodeList [ ]: Contains a list of nodes to which the Result needs to be copied.
            At Node $N_i$
for each node $N_j$ in NodeList [ ]
If $E_{ij}$ exists; $E_{ij}$ is the edge between $N_i$ and $N_j$
M=11
Transfer Result from $N_i$ to $N_j$
If $P_{ij}$ exists; $P_{ij}$ is the path between $N_i$ and $N_j$
            M=10
Transfer $Res_1$ from $N_i$ to next node in path $P_{ij}$
else
COPY cannot be executed for node $N_j$
At node $N_k$;can be intermediate node or destination node
If DestID=SelfID copy $Res_1$.
If DestID≠SelfID transfer $Res_1$ to next node in $P_{ij}$
```

7.1 Message exchange for adjacency list maintenance at each node

Each node to indicate its presence in the network sends periodic **ALIVE** message to those not connected to it. This message is essentially a multicast message. The flag **A** is set to indicate that the message is an **ALIVE** message. A node's absence is determined due to the missing **ALIVE** message and changes are made in the connectivity graph of each of the neighbors.

Also, a node can voluntarily leave a network by sending a **QUIT** message (also multicast) and hence changes are made in the connectivity graph of each of the neighbors. The flag **Q** is set to indicate that the node is voluntarily leaving the network. A typical sequence of message exchanges between two nodes in shown in Figure 13.

Algorithm to reconstruct the Adjacency list

RECONSTRUCT_ADJLIST

```
{       At node $N_i$
        If no ALIVE message is received from Node $N_j$ or QUIT is received from $N_j$
            Remove node $N_j$ from the adjacency list of node $N_i$
            For each node $N_k$ in the adjacency list of node $N_i$
                N=1
                Send NEIGHBOR message (updated adjacency list, Meta data
            information)
}
```

Sequence of message exchange between nodes in the network.

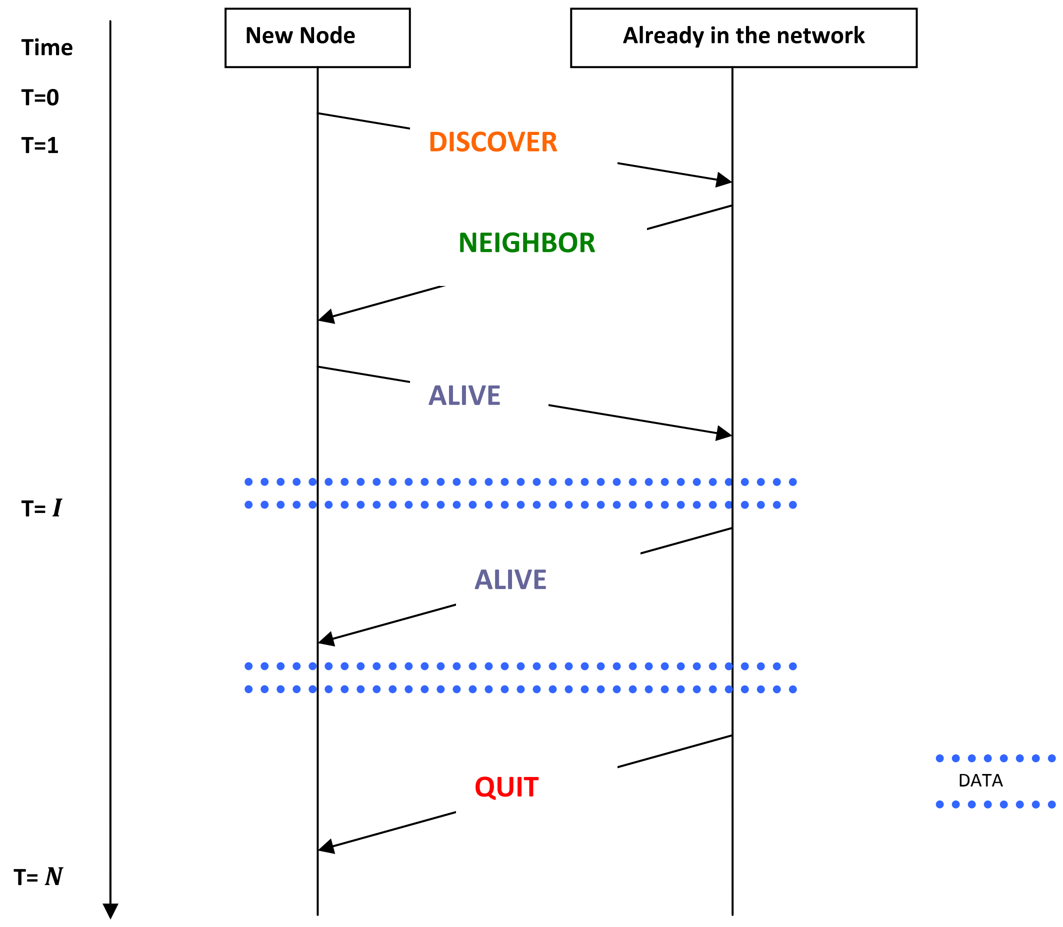


Figure 13: Message exchange between nodes from T=0 to T=*N* units

7.2 Replication

Each data object is replicated on other node. In the following we discuss the criteria for choosing an appropriate node for replication.

Assumptions:

There exists only one replica of a given data item (Tuples).

$\boldsymbol{N_s}$(Source Node): The node where the original copy of data item $\boldsymbol{d_i}$ exists.

$\boldsymbol{N_c}$(Candidate Node): The node which contains the replica of $\boldsymbol{d_i}$.

When a Source node $\boldsymbol{N_s}$ decides to replicate its contents on another node $\boldsymbol{N_c}$, a node from the set of the Nodes that are immediate neighbors of the source node is selected as candidate node for replication. Immediate neighbors are those nodes which are directly connected to the source node. The source node tries as much to replicate all its tuples on the chosen candidate node.

For each of the above selected candidate nodes, a cost function $\boldsymbol{C_{s,c}}$ is computed. The node with the lowest cost is selected as a candidate for replication.

The cost function to determine the candidate node for replication is dependent on the following factors:

Bandwidth: Defines the closeness of $\boldsymbol{N_c}$ from $\boldsymbol{N_s}$ in terms of bandwidth. Greater the bandwidth more data can be transmitted within a given time. This is advantageous when there are several tuples to be sent across the link in a single update operation.

Link stability: This is a measure of stability of the link between nodes $\boldsymbol{N_s}$ and $\boldsymbol{N_c}$. Greater stability of the link between the two nodes implies better longevity.

The **Link stability** $\boldsymbol{LS_{s,c}}$ measured at $\boldsymbol{N_s}$, is equal to rate of change of received signal strength for messages sent by $\boldsymbol{N_c}$. Let $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ be the received signal strength at time $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ respectively. ($\boldsymbol{t_2}>\boldsymbol{t_1}$)Current Link stability between $\boldsymbol{N_s}$ and $\boldsymbol{N_c}$, is given by,

$\boldsymbol{LS_{s,c}} = \frac{dr}{dt}$,Where $\boldsymbol{dr = r_2 - r_1}$ and $\boldsymbol{dt = t_2 - t_1}$. If $\boldsymbol{r_2}>\boldsymbol{r_1}$ then $\boldsymbol{LS_{s,c}}$ is positive, this indicates increasing link stability between $\boldsymbol{Ns}$ and $\boldsymbol{N_c}$. If $\boldsymbol{r_2}<\boldsymbol{r_1}$ then $\boldsymbol{LS_{s,c}}$ is negative, this indicates decreasing link stability between $\boldsymbol{Ns}$ and $\boldsymbol{N_c}$.

Degree of the node $\boldsymbol{N_c}$: Greater the degree of a node, better is the accessibility of replicated data.

7.2.1 Cost function

When the source node $\boldsymbol{N_s}$ decides to replicate its data item $\boldsymbol{d_i}$ on a candidate node

Table 11: Cost function parameters maintained at each node N_s

Node	**Average Bandwidth**	**Average Degree**
N_1	B_1	D_1
N_2	B_2	D_2
N_i	B_i	D_i

$\boldsymbol{N_c}$, it takes into account the recent history (from the last replication decision) to make the replication decision. History includes information about the above mentioned three factors. A simple average is computed for each of Bandwidth and Degree. A table, such as Table 11 is maintained to store the average values for each node the source node is connected to. Each row in the above table can be considered as a vector of values. Each element in this vector is normalized and has a value between 0 and 1.

For each candidate node, a table is maintained to store the RSS values recorded in time. At any point in time, the Link stability can be determined using 2 consecutive values from the Table 12. ($t_1<t_2<t_i$)

Table 12: RSS values

Time	**Relative Signal Strength**
t_1	r_1
t_2	r_2
t_i	r_i

Consider the following example:

Let the current value of Bandwidth between nodes N_1 and N_2 be 500 kpbs and the maximum value of bandwidth be 1Mbps. The normalized value of bandwidth is 500kbps/1Mbps, which is equal to 0.5. Similarly, normalization is applied to degree.

For link stability normalization is applied as follows:

> If the node $\boldsymbol{N_s}$ decides to replicate it's tuples at say time $\boldsymbol{t_i}$, it takes into account the current value of RSS $\boldsymbol{r_i}$ (time $\boldsymbol{t_i}$)and the previous value $\boldsymbol{r_{i-1}}$(time $\boldsymbol{t_{i-1}}$).The current Link stability of the link

between $\boldsymbol{N_s}$ and $\boldsymbol{N_c}$ is equal to the normalized value of rate of change of RSS. Let '***T***' be the time period between the last replication decision and the current replication decision. The maximum value of rate of change of RSS during '***T***' is computed which is used in the normalization.

Link stability $\boldsymbol{LS'_{s,c}}$: $\dfrac{dr\,/dt}{MAX\left(\dfrac{dr}{dt}\right)}$

The above value lies in the range [-1, 1].

Normalized link stability is given by: ($\boldsymbol{LS'_{s,c}}$+α)/β where α=1 and β=2.The normalized value has a range [0,1].

After each parameter is normalized, a weight is associated with it. The weight is an indication of the relative importance of a parameter with respect to other parameters in the computation of the cost function. Let $\boldsymbol{w_b}$, $\boldsymbol{w_l}$ and $\boldsymbol{w_d}$ be the weights assigned for each of average bandwidth, link stability and average degree respectively.

Hence the cost function $\boldsymbol{C_{s,c}}$ to identify the most suitable node for replication is given by,

$\boldsymbol{C_{s,c}}$ =1/($\boldsymbol{w_b}$*average bandwidth + $\boldsymbol{w_l}$*link stability + $\boldsymbol{w_d}$* average degree)

Also, $\boldsymbol{w_b + w_l + w_d} = 1.$

Lower the value of $\boldsymbol{C_{s,c}}$ for a candidate node $\boldsymbol{N_c}$,indicates it is more suitable for replication.

Weights for $\boldsymbol{w_b}$, $\boldsymbol{w_l}$ and $\boldsymbol{w_d}$: The range of values for each of $\boldsymbol{w_b}$ and $\boldsymbol{w_l}$ is [0,0.4]and $\boldsymbol{w_d}$ is [0,0.1].

Algorithm to replicate

A node decides to replicate its tuples after each time period *'T'*.

Algorithm to Check Connectivity

```
CONNECTIVITY (connectivity matrix, $N_s$)
{       for each Node $N_j$ in connectivity matrix of node $N_s$
If $E_{sj}$ exists; $E_{sj}$ is the edge between $N_s$ and $N_j$
Return $N_j$
Else
Return null
}
```

Justification for the parameters used in the cost function.

The cost function is given by:

$\boldsymbol{C_{s,c}}$ =1/($\boldsymbol{w_b}$*average bandwidth + $\boldsymbol{w_l}$*link stability + $\boldsymbol{w_d}$* average degree)

The network in this scenario is dynamic and the connectivity between nodes (physical nodes) keeps changing very often. The above chosen parameters in the cost function are ideal to select a replica.

Bandwidth:

During the process of replication, depending on how much data the source node has acquired, the amount of data that needs to be transferred (during a replication period) to the candidate node can be a lot. Also, taking into consideration the dynamic nature of the network, delays in transmission can lead to some data not being transferred to the candidate node. If the candidate node is queried (instead of the source node) the result of the query may not be accurate due to lack of tuples necessary for carrying out the execution of the query. Bandwidth plays an important role in transmission delay, since greater the bandwidth more data can be transmitted on the link within less time. If the source node and the candidate node are connected and the bandwidth between them is high, data can be replicated quickly on the candidate node.

Degree:

If the source node of a data item goes out of range (isolated) for a long period of time, any query involving data on source node cannot be successfully executed. If a node '$\boldsymbol{N_c}$' is connected to several other nodes, data on '$\boldsymbol{N_c}$' will be available to other connected (direct or indirect) nodes for a longer period of time. It is essential that the replica of a data item (tuples) be accessible when the actual source of data goes out of range. If the candidate node has a low degree then the chances of it going out of range is greater. Hence degree of a node plays an important role in the availability of data.

Link Stability:

Link stability is a measure of a source node's connectivity with the candidate node. Greater the Link stability between 2 nodes, the more stable is the connectivity between the two nodes. A source node '$\boldsymbol{N_s}$' always replicates its data items (tuples) on a single candidate node '$\boldsymbol{N_c}$'. If the connectivity between '$\boldsymbol{N_s}$' and '$\boldsymbol{N_c}$' keeps changing very often (highly dynamic) and '$\boldsymbol{N_s}$' has acquired new data which it decides to replicate on '$\boldsymbol{N_c}$', replicating data becomes hard and unmanageable. If the link stability between 2 nodes is high, the source node gets more time to replicate its data. When a query hits a candidate node, it will have the most recent data (since the source node can continuously transmit the recently acquired tuples) required for successful query execution.

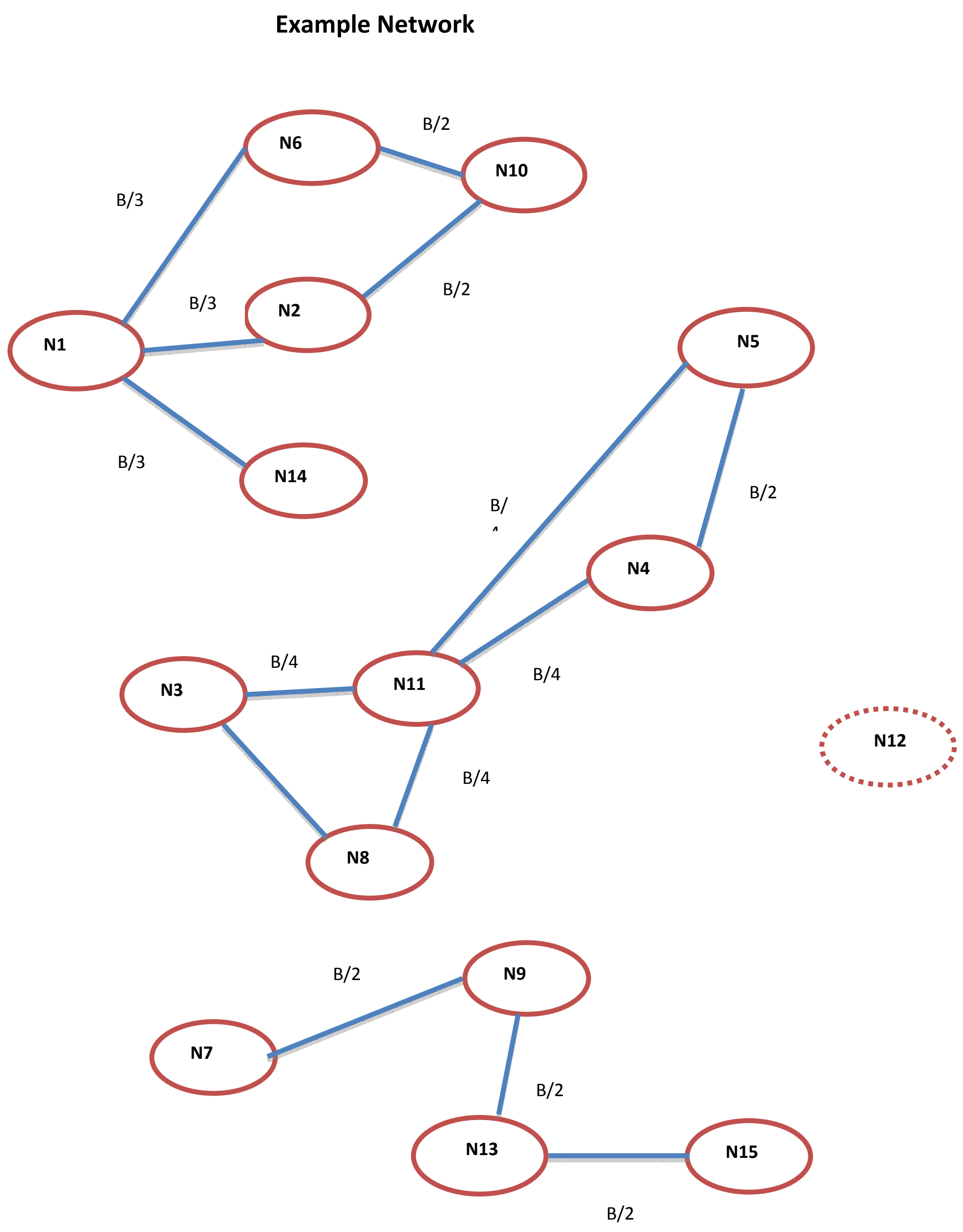


Figure 14 A sample graph of 14 nodes in the system

B = 1 Mbps, B/2=500Kbps, B/3=333kbps, B/4=250kbps

Plots

W_l =0.4, W_b =0.4, W_d=0.2

In the above graph if Node N1 decides at t6 to replicate its tuples on one of N2, N6 or N14, the best node in terms of cost (lowest) is chosen for replication.

The cost calculation for each of these nodes is shown below:

Cost for replicating on node N2

Table 13 RSS Values from t1 to t5 for Node 2

t1	t2	t3	t4	t5	t6
5	7	4	8	6	5

(Maintained at N1)

Table 14 Parameter values for Node 2

	Current Value	Maximum Value	Normalized value
Bandwidth	333kbps	1Mbps	0.33
Degree	2	14	0.14
Link stability	-1	4	0.37

Cost = 1/ ((0.33*0.4) + (0.37*0.4) + (0.14*0.2))=1/0.30=**3.33**

Cost for replicating on node N6

Table 15 Parameter values for Node 6

	Current Value	Maximum Value	Normalized value
Bandwidth	333kbps	1Mbps	0.33
Degree	2	14	0.14
Link stability	1	3	0.66

Table 16 RSS Values from t1 to t5 for Node 6

t1	t2	t3	t4	t5	t6
5	7	6	4	7	8

(Maintained at N1)

Cost = 1/ ((0.33*0.4) + (0.66*0.4) + (0.14*0.2))=1/0.42=**2.35**

Cost for replicating on node N14

Table 17 Parameter values for Node 14

	Current Value	Maximum Value	Normalized value
Bandwidth	333kbps	1Mbps	0.33
Degree	1	14	0.07
Link stability	-1	3	0.33

Table 18 RSS values from t1 to t6 for Node N14

t1	t2	t3	t4	t5	t6
5	7	6	4	7	6

(Maintained at N1)

Cost = 1/ ((0.33*0.4) + (0.33*0.4) + (0.07*0.2))=1/0.27=**3.70**

Table 19: Cost for nodes N2, N6 and N14

	Bandwidth	Degree	Link Stability	Cost
Node N2	0.33	0.14	0.37	3.33
Node N6	0.33	0.14	0.66	2.35
Node N14	0.33	0.07	0.33	3.7

From the above table, Node N1 chooses node N6 as the best node for replication since the cost for replication is lesser than the other 2 nodes.

7.3 Application Level mechanism for Acknowledging UDP packets

Issues with TCP over mobile wireless network: The nodes in the scenario communicate with each other over wireless links. TCP does not scale well such wireless networks as nodes frequently disconnect and rejoin. It will be necessary to use UDP for transfer of messages/metadata between nodes in the current environment. However, for critical applications, reliable transfer of packets is imperative. In view of the above, we have developed a simple mechanism for acknowledging UDP packets from the application layer. Development of the simple mechanism was preceded by a review of existing literature on the topic. Summaries of critiques will be provided if needed.

1. Wireless links use open air as the transmission medium and are subject to many uncontrollable quality-affecting factors such as weather conditions, urban obstacles, multipath interferences, large moving objects, and mobility of wireless end devices. As a result, wireless links exhibit much higher Bit Error Rate (BERs) than wired links.

2. Limitations of radio coverage and node mobility require frequent handoffs, thus resulting in temporal disconnections and reconnections between the communicating end hosts during a communication session. During a disconnection both the retransmission and ACK's will be lost, repeated failed retransmissions results in stalling the TCP transmission for a much longer period. (Slow start)

UDP data transfer scheme:

The User Datagram Protocol provides a procedure for application programs to send data to other applications with minimum protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. However UDP can be used to achieve high speed as well as accurate results by providing an acknowledgement scheme in the application layer. Unlike the TCP protocol where the acknowledgement is sent after each packet is sent, by using the UDP scheme the acknowledgement for the data received can be sent only after a block of data is received. This reduces the number of acknowledgements sent when compared to the TCP scheme hence increases the overall speed.

The structure of a UDP frame includes:

Source Port	Destination Port
Length	Checksum
Data Octets	

Application layer acknowledgement scheme based on UDP:

In the application layer at the sender side, an **INFO** message which contains information about the number of packets and the number of tuples is sent to the receiver before the actual data is transmitted.

The format of the **INFO** message is:

SP	F	D	N	I	M	A	Q	Source ID	Destination ID	Timestamp	Payload

The flag **F** is set (**F=1**) to indicate that the payload contains frame related information. The payload contains information about the number of packets that would follow ('K') and the number of tuples in each of the packets ('k'). This information lets the receiver know about the amount of data to expect from the sender identified by the Source ID field. The receiver allocates some buffer space to accommodate the incoming data.

After the **INFO** packet has been transmitted, the sender transmits the actual data. Each application layer data packet (those which have the **M** flag set) contains information about the **ID** for the transaction and the sequence number "**Seq**" of the packet being sent. The **ID:Seq** field for the other packets has a value of 0 to indicate that the message contains control packets.

SP	ID: Seq	F	D	N	I	1	A	Q	Source ID	Destination ID	Timestamp	Payload

The receiver maintains 2 timers ACK and NAK which are for acknowledging and reporting loss respectively. These indicate the frequency of acknowledgment (or NAK) in the network. If the ACK timer expires and there are new packets to be acknowledged, the receiver sends an ACK to the sender. The receiver maintains a loss list which maintains information about the number of packets that were lost during transmission. If the current sequence number is greater than the largest sequence number ever received plus 1, all the sequence numbers between these two numbers will be inserted into the receiver's loss list. If the NAK timer expires and the loss list at the receiver is nonempty then the receiver sends a NAK to the sender.

On receiving a NAK, the sender retransmits the packet. Until an ACK is received the sender maintains information about all the packets that have been sent in this session (indicated by **ID**) in the sender's ToBeACK list. On receiving acknowledgement report for a particular packet (or set of packets) from the receiver, information about these packets is removed from the ToBeAck list.

The format of the **NA** packet is:

NA	ID: Seq	F	D	N	I	M	A	Q	Source ID	Destination ID	Timestamp	Payload

The field **NA** is set to **10** to indicate a loss report. The payload contains information about the packet numbers not yet received. The field **NA** is set to **11** which indicate the acknowledgment till the last correctly received packet.

<u>Algorithm for sending data (at the sender):</u>

The algorithm “Send” transmits data “Data” to the receiver “Destination ID”.

```
Send (Data, Destination ID) {

Send control information by setting the flag F=1 of the INFO message to the receiver.

Send data (tuples) using UDP to receiver (Destination ID)

Add the ID: Sequence number in the ToBeACK list.

On receiving an ACK, remove the Sequence number (s) from the ToBeACK list.

On receiving a NAK, retransmit the packet(s) to the receiver.

}
```

<u>Algorithm for receiving data (at the receiver):</u>

The Receive algorithm is executed at the receiver.

```
Receive (Data, Sender ID, Destination ID) {

After receiving the INFO message, start the ACK and NACK timer.

Receiving packet: if Sequence number > Sequence number in buffer +1

        Add packet to loss list.

If NAK expires and loss list is nonempty

                Send NAK report to sender

If ACK expires and there are pending acknowledgements to be sent to the Sender

                Send an ACK report to the sender.
```

8. SIMULATION STUDIES

8.1 Simulation Architecture

Technical Specifications:

Table 20 Specifications for Simulation

MODEL	**TECHNOLOGY**
Front end	JAVA AWT
Back end	ORACLE 10G
Controller	JDBC
Languages	JAVA, PL/SQL
SDE	ECLIPSE

8.1.1 Description or Setup:

A mobile Peer-to-Peer network has been set up with a maximum of 12 UAV's. All the UAV's are within an area of 140 X 140 miles (area of 20,000 sq. miles) distance.

Each UAV sends a DISCOVER message as soon as it enters the network. UAV's within the transmission range receive this DISCOVER message and acknowledge it by sending a NEIGHBOR message back. A links is then created between these UAV's. UAV's also keep sending DISCOVER message periodically to find new UAV's reachable.

An ALIVE message is sent periodically by all the UAV's to indicate their presence in the network. An ALIVE message also contains the connectivity graph of the UAV.

Bandwidth of the link is 1Mbps if both the UAV's have only one link; it is 500kbps if at least one UAV has 2 links and so on. The bandwidth of the link also depends on the length of the link (Distance between UAV's). The maximum bandwidth is 1Mbps till a distance of 30 miles and bandwidth decreases to 180 kbps as the distance increases to 60 miles.

UAV "0" acts as the Broker node or Root node. UAV's which have a direct link to UAV "0" are called Level 1 nodes. The remaining UAV's are level 2 nodes or normal nodes.

8.1.2 Links in the Network*:*

Each UAV tries to create a link with all other UAV's within its area. The bandwidth then depends on the number of links present at both the nodes. Since UAV '0' is a root node, we allow only 3 connections for UAV '0'. All other UAV's can have any no. of links.

8.1.3 *Movement of UAV's*

The speed of all the UAV's is set to 100 nautical miles/hour.

Random Way point Model: UAV's move around within the area randomly. Random Way Point model has been implemented for the movement of UAV's. As the UAV's move, bandwidth of the links change based on the distance between them. Also links are broken and new links formed based on the distance. UAV's may not move always. Sometimes they stay at fixed position and this also has been simulated.

Fixed Trajectory: Each UAV is assigned a fixed path when it is created. UAV's can be asked to move in this path. Fixed path trajectory has been implemented for it. The speed for this fixed trajectory is also the same. UAV's either move randomly or along the path fixed. Both of them are not possible at the same time.

Circular movement of UAVs: On addition to the fixed path trajectory having a polygon of any random shape, we have incorporated the circular fixed path trajectory also. The current system has both circular path fixed trajectory and polygon shaped fixed path trajectory. The trajectory can be viewed in the simulation environment by entering the node label in the text box beside show path and clicking show path.

8.1.4 *Routing Algorithm*

Since the number of UAV's are less, a two layer routing protocol has been implemented. For this each UAV broadcasts information about its neighbors and the distance between them (i.e. each UAV broadcasts its neighbor table). This is done by sending an ALIVE message periodically.

Using all the received ALIVE messages, each UAV has an overview of the graph (length of edges and UAV's) in the first layer. The graph is stored in the memory of the UAV and is updated as soon as ALIVE messages are received. So each UAV has the most recent possible graph structure.

To also reduce the time taken to find a route between UAV's, each UAV finds a route to all other UAV's and stores it in the second layer. These routes are found from the graph in the first layer using Dijkstra's algorithm.

Whenever an existing link is broken, the UAV's involved send an ALIVE message again. This is done to update the graph structure at every UAV. UAV's can then decide whether to find a route again or to use an existing one.

When a new link is formed, the UAV's do not broadcast data about the new link. This is because as soon as the time period expires each UAV broadcasts an ALIVE message again, so the new link information is also broadcasted. Since we are not sure if this link will help decrease the cost incurred by a considerable level, we wait for the timer to expire rather than broadcasting about the new link again.

8.1.5 *Orientation*

We have each UAV rotating at four specific angles (0, ∏/2, ∏, 2∏). We assume that each UAV will be rotating to any one of these angles for every one minute. We then reduce the initial available bandwidth based on distance by subtracting it with a reduction factor. The reduction factor depends on the orientation of the UAV's, there are 16 such cases for four angles as discussed in the **Table 21** below. We are categorizing UAV's as UAV1and UAV2 based on their location in the simulation area.

Let UAV **(v),** UAV **(w)** be **any two** UAV's.

Case 1: If (X axis value of UAV (v) > X axis value of UAV (w)) UAV1 = UAV (w) & UAV2= UAV (v)

Else UAV1 = UAV (v) & UAV2= UAV (w); Shown in Figure 15.

Case 2: If (Y axis value of UAV (v) > Y axis value of UAV (w)) UAV1 = UAV (w) & UAV2= UAV (v)

Else UAV1 = UAV (v) &UAV2= UAV (w); Shown in Figure 16.

The ideology of using **case 1** is to determine the left and right UAV's in the simulation area. Case1 ensures that UAV1 will be always to the left of UAV2.

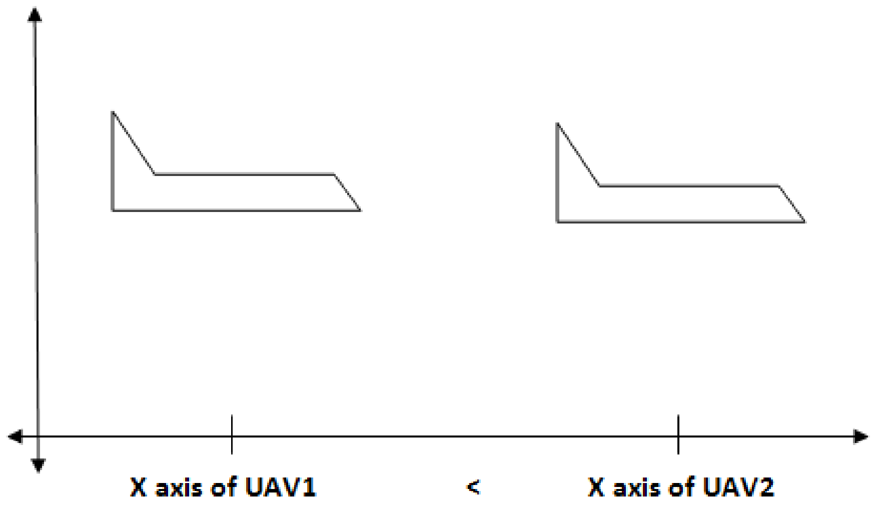


Figure 15 Relative positions of UAVs: Case 1

There are cases in which the X axis of the UAV might be the same, i.e.) they might be lying in the same vertical axis. Hence, we have case 2 determining top and bottom UAV's in the simulation area. Case 2 ensures that UAV1 is always to the bottom of UAV2.

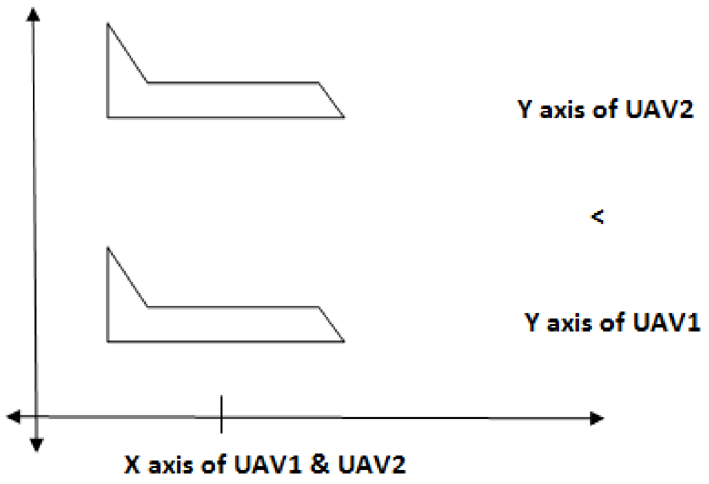


Figure 16 Relative positions of UAVs: Case 2

Table 21 Reduction factors for the two UAVs

NO	UAV1	UAV2	REDUCTION FACTOR Case 1	REDUCTION FACTOR Case 2
1	0	0	5k	5k
2	∏/2	0	6k	3k
3	∏	0	7k	4k
4	3∏/2	0	6k	6k
5	0	∏/2	3k	6k
6	∏/2	∏/2	5k	5k
7	∏	∏/2	6k	6k
8	3∏/2	∏/2	4k	7k
9	0	∏	0	4k
10	∏/2	∏	3k	3k
11	∏	∏	5k	5k
12	3∏/2	∏	3k	6k
13	0	3∏/2	3k	3k
14	∏/2	3∏/2	4k	0
15	∏	3∏/2	6k	3k
16	3∏/2	3∏/2	5k	5k

Constant **k**= 6

Bandwidth after orientation=Original Bandwidth - Reduction Factor

The reduction factor is designed by assuming the antenna to be in front of the UAV's. For e.g., in row number 9 of case 1, UAV1 is 0 degree and UAV2 is 180 degree, since both UAV's are facing each other it will get the maximum bandwidth, as shown below in Figure 17.

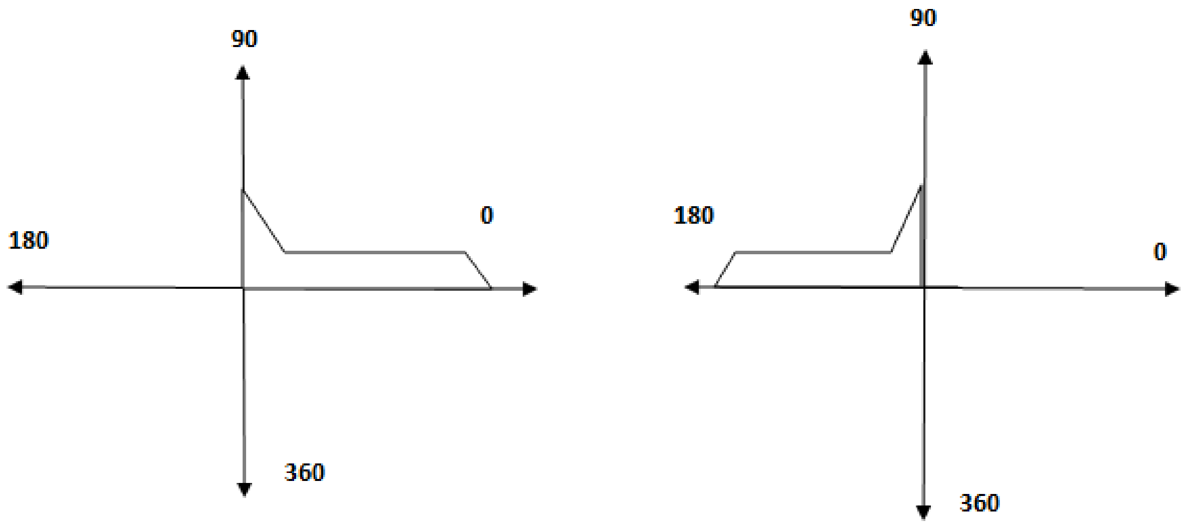


Figure 17 UAVs facing each other

Whereas, if we consider row number 3 of case 1, UAV1 is 180 degree and UAV2 is at 0 degree; both UAV's are facing opposite to each other as shown below, hence this event bears the highest reduction factor.

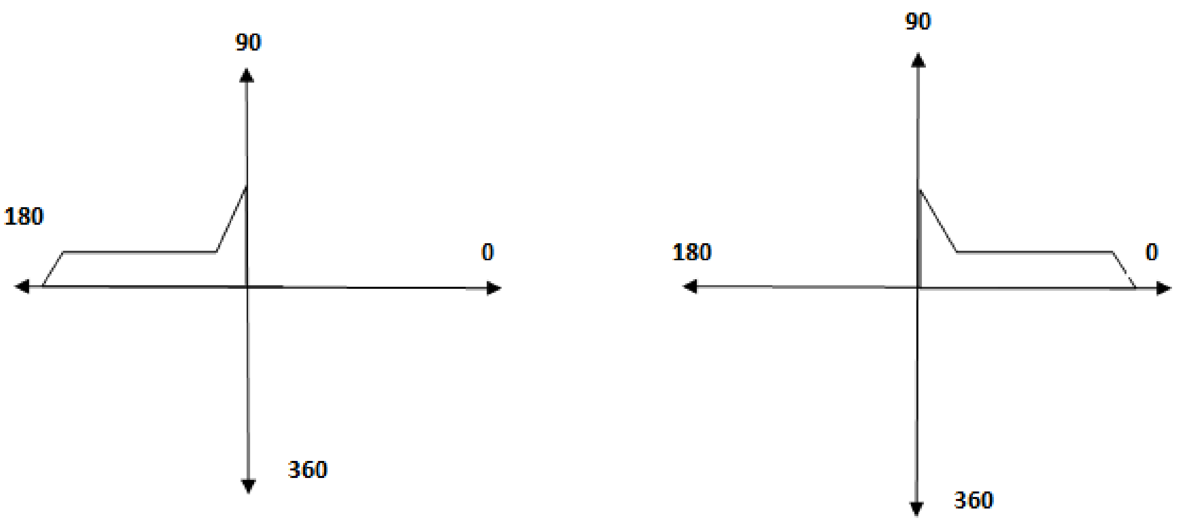


Figure 18 UAVs facing in opposite directions

8.2 NETWORK CONFIGURATION & MESSAGE PROTOCOL

8.2.1 ASSUMPTION

All UAV's (also called as nodes here) are homogenous and they each carry a device that has capabilities similar to a laptop and that has support for wireless communication. The number of nodes ranges from 10-16. The message format for the various messages exchanged is similar and it contains multiple flags to indicate the type of data being sent in the payload. The address of each UAV is of length 5 bits and it is unique to each UAV.

Msg. Id	**D**	**A**	**L**	**M**	**S**	Source ID	Destination ID	Timestamp	Payload

D-Discover
A-Alive
L-Link
M-Message/Data
S-Status
Spare Flags, reserved for future use

8.2.2 SETUP

Initially there is a single node that exists in the network.

At say time T=t1 there is a node which comes into the range. The new node sends a neighbor discovery DISCOVER message. The DISCOVER message contains

- The source identifier (ID).
- The destination ID is typically a Broadcast address.
- A Timestamp.
- A list of data it contains.
- The Flag **D** when set indicates that the payload contains information about the data on that node.

If the node is the only one in the network for the current time slot, it sends up to 'N' neighbor DISCOVER messages and decides that it is the only one in the network. The DISCOVER messages are sent periodically to gain new connections. Say at time T=t2, there are other nodes which come into the range. Each of these nodes send a similar DISCOVER message. When an already existing node receives a DISCOVER message, if the node is within a distance of 40 miles and if it has free channels available, it responds with an acknowledgement. A node does this by sending an ALIVE message. The ALIVE message contains:

- The source identifier (ID).
- The destination ID (which has previously sent the DISCOVER message).
- The connectivity graph / Adjacency matrix.
- The Flag **A** when set indicates that the payload contains information about the nodes connectivity.

If all the nodes in the network have 3 connections (i.e. if all the channels are taken up), one node is chosen to form a link. This node is found from the set of nodes within a distance of 30 miles from the node sending DISCOVER message. Nodes farther than 30 miles do not reply to the DISCOVER message. The selected node breaks an existing link by sending a LINK message. It then replies with an ALIVE message to the node sending DISCOVER message. The LINK message contains:

- The source identifier (ID).
- The destination ID (the node to which link is broken)
- A Timestamp
- Flag L is set.

If a node is many hops away from any node, LINK message can also be used to break an existing link and create a new link. This is done so that a least used link is broken and a link is created between nodes which are many hops away. The least used link is found from the routing table. Weight is assigned to each link based on the number of nodes it can reach using that link. The link with least weight is broken to create a new link.

If the node receiving LINK message has a link with the source, that link is broken. If the node receiving LINK message does not have any link with source and if the nodes are within a distance of 30 miles, a new link is created with the source. A new link is acknowledged by sending an ALIVE message back. After this initial setup phase each node in the network has information about both the connectivity as well as the data that each of its neighbors nodes possess. The connectivity information helps in routing. Routing is required for those nodes which want to send data to other nodes which are not directly connected.

8.2.3 CONSTRUCTION OF GN

Each node broadcasts a ALIVE message periodically. This ALIVE message also contains the neighbor table like structure for each node. This table contains the nodes it can connect to and the distance to each of them.

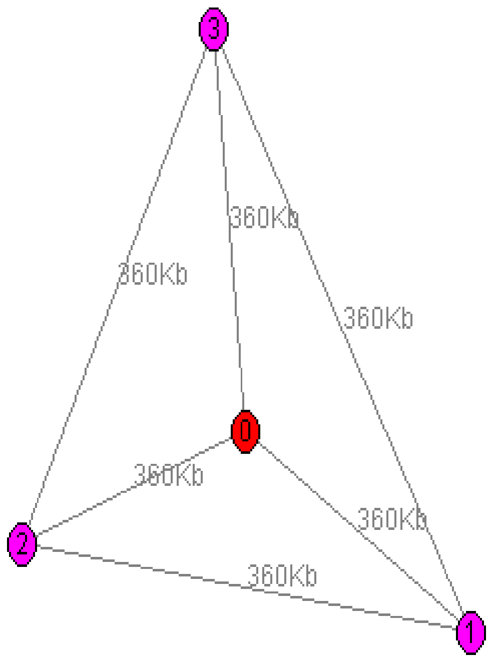


Figure 19 Graph representing 4 nodes

From Figure 19, information broadcasted by node 1 is:

1	0	Distance between 0 and 1
1	2	Distance between 1 and 2
1	3	Distance between 1 and 3

Each node broadcasts such information to the whole network using the links. An adjacency matrix with the distance is created at every node from this information. This adjacency matrix is used to find the shortest route to the nodes in the network. The shortest route is found using Dijkstra's algorithm on the adjacency matrix. At any point of time, there are 18 links in the network with 12 nodes. So the amount of link information sent is 18 lines. The routing table thus formed is used to find the least used link and also to check the connectivity in the network. The routing table for a network is stored in the following format in Table 22.

Table 22 Routing table

```
Routing Table for 2 :
 2 4 7 0
 2 11 1

 2 4 3
 2 4
 2 4 7 5
 2 6
 2 4 7
 2 4 3 8
 2 11 9
 2 4 7 10
 2 11
```

Row 0 shows the shortest route from node 2 to node 0.
Row 1 shows the shortest route from node 2 to node 1 and so on.
If any row is empty, then the node is not able to reach that node. It tries to connect to that node by sending LINK message to that node.
If the no. of hops to any node is more, (for e. g., if the 10th row is 2 4 7 0 1 10--- six hops) node 2 tries to establish a direct connection with node 10 if it is within reachable distance. The link to be broken is decided based on the weights. The weights of the links are:
Link 2-4 --- 7
Link 2-6 --- 1
Link 2-11--- 3
Since the link 2-6 has less weight, this link is broken to form a new link with 10.
Each node to indicate its presence in the network sends periodic ALIVE message to those who are connected to it. This message is essentially a broadcast message. This helps in maintaining the updated distances and bandwidth of the links. With this algorithm, information about all the links is known to all the nodes in the network. This helps in finding the best shortest path in the network. In the previous algorithm, the information is only shared with the neighbors. The neighbors then share this information in the next cycle. This increases the time to update the table at the other end of the network. This network can also change by the time information reaches other end. Using this mechanism, the updated information is known to all the nodes and helps in finding the best route every time.

8.2.4 RECONSTRUCTION OF GN

If no ALIVE message is received from a node, that node is considered to be disconnected. The neighboring nodes then broadcast the routing table again and also broadcast a DISCOVER message. DISCOVER message is broadcasted because there is free channel available.

Whenever an ALIVE message is received by any node, the distance between the nodes is updated and the shortest route to every node is found again.

8.2.5 GRAPH MAINTENANCE

Graph structure of the network is maintained by storing the distance between the nodes. This distance is updated whenever a ALIVE message is received. The ALIVE message is broadcasted under these conditions:

1. When acknowledging a new link
2. Periodically to update the distance between nodes.
3. Whenever a link is deleted from the network.

8.3 METADATA GENERATION

The metadata information is stored at each node. Every node will broadcast its MIO information. Every other node computes the metadata information for data present at other nodes using the received MIO information. The metadata information metrics are cardinality, Width, Min value and Max value. These metrics are used for computing cost during query plan generation. The metadata table with metadata information metrics is shown in Table 23.

We are using **Triggers** to update the metadata metrics, when new data are inserted into the MIO table.

Object Type	Node ID	Size	Object Description	Latitude	Longitude	Time Stamp	Received Time Stamp

MIO TABLE ATTRIBUTES

Table23: METADATA TABLE

Name	**Type**	**card**	**width**	**Min value**	**Max value**	**Range (unique values)**
Object Type	Rel					
Size	number					
Object Description	Varchar					
Latitude	number					
Longitude	number					
Time Stamp	Varchar					

8.4 REPLICATION

There exists only one replica of any data item prevailing in the system. We have the source node containing the original data item and one of the candidate nodes containing the replica. The candidate nodes are one hop neighbors of the source node; hence replication is restricted to one hop. Each node will have many candidate nodes; hence the choice of replicating a data item at a candidate node is based on a cost function comprising **Bandwidth, Link Stability and Degree of Node.**

$\mathbf{W_b, W_l, W_d} \leftarrow$ Weights assigned for bandwidth, link stability and degree of node.

$\mathbf{Cost_{s,c} = 1}(W_b$*average bandwidth + W_l* link stability + W_d * average degree)

Once a node replicates its data to one of its neighbors, it places the replica information in the replication mapping table. The replication mapping table will be maintained at each node. The replication mapping table contains information about name of the data item being replicated, original node and the replicated node.

8.5 STATIC QUERY PLAN EXECUTION

On an environment at which partial meta-data is broadcasted, i.e. Meta data containing only data_desc and source information, we have static query plans to evaluate queries requiring other meta-data information such as latitude, longitude etc. A query plan is a numbered sequence of steps that can be easily interpreted and executed at any node as shown below.

Operation 1	Param	Operand1	Operand1 Loc	Operand2	Operand 2 Location	Result Name	Result Loc
Operation 2	Param	Operand1	Operand1 Loc	Operand1	Operand2 Loc	Result Name	Result Loc
…	…	…	…	…	…	…	…
Operation n	Param	Operand1	Operand1 Loc	Operand1	Operand2 Loc	Result Name	Result Loc

The query execution proceeds as follows. The plan is sent to the node in which the first operation takes place. The interpreter in that node uses the plan counter to execute as many steps as possible in that node. When a move is encountered, it sends the meta-data results as well as the plan to the node mentioned in Result Loc of the plan. This process continues until the last step of the plan is executed. The final node containing meta-data result retrieves the needed Data either from replica or original node.

We have the following set of operations SELECT, MOVE and JOIN; Param indicates the condition for select and join operations. Operand1 & Operand2 are variables used for join operation. The result of select and join operation are placed in the Result Name variable.

*"SELECT * FROM UAV1 R_1, UAV2 R_2, UAV3 R_3 WHERE R_1.timestamp LIKE k AND R_2.timestamp LIKE j AND R_3.Obj_type='1' AND R_1.latitude>R_2.latitude AND R_2.longitude>R_3.longitude"*

k, j ← Input timestamps given by the user.

The query plan for the above query is shown in the table 24. The query plan is given to node 0 by default, which follows execution as per the plan.

Table 24: Query plan

Operation	Sub SQL & JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT LATITUDE FROM UAV1 WHERE TIME_STAMP LIKE k	R_1	1	Null	Null	R_1'	1
Move	Null	R_1'	1	Null	Null	R_{12}	2
Select	SELECT LATITUDE,LONGITUDE FROM UAV2 WHERE TIME_STAMP LIKE j	R_2	2	Null	Null	R_2'	2
Join	Latitude > Latitude	R_{12}	2	R_2'	2	J_1	2
Move	Null	J_1	2	Null	Null	J_{12}	3
Select	SELECT * FROM UAV3 WHERE OBJ_TYPE=1	R_3	3	Null	Null	R_3'	3
Join	Longitude > Longitude	R_3'	3	J_{12}	3	J_2	3
Move	Null	J_2	3	Null	Null	J_{37}	7

We also simulated query plans for the following scenarios.

1. Get all information from UAV taken within last five minutes of the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

*SELECT * FROM UAV1 R_1, UAV2 R_2 WHERE R_1.timestamp >time AND R_2.timestamp < time AND R_1.timestamp > time AND R_2.timestamp < time AND R_1.latitude>R_2.latitude AND R_1.longitude>R_2.longitude*

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1 WHERE R1.time_stamp >to_date('08-dec-2010 09:41:30 PM','dd/mm/yyyyhh:mi:ss PM') AND R1.time_stamp < to_date('08-dec-2010 09:41:32 PM','dd/mm/yyyyhh:mi:ss PM')	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2 WHERE R2.time_stamp >to_date('08-dec-2010 09:41:30 PM','dd/mm/yyyyhh:mi:ss PM') AND R2.time_stamp < to_date('08-dec-2010 09:41:32 PM','dd/mm/yyyyhh:mi:ss PM')	R_2	2	Null	Null	R_{21}	2
Join	Latitude> Latitude and Longitude > Longitude	R_{11}	2	R_{21}	2	J_1	2
Move	Null	J_1	2	Null	Null	J_1	3

The date given in the query and query plan are sample dates, our simulation run with respect to current time. It retrieves data taken by UAVs since last 5 minutes.

2. Get all images taken within last five minutes of the area bounded by latitude1, longitude1 and latitude2, longitude2 within 2 NM, 3NM.

Query:

*SELECT * FROM UAV1 R_1, UAV2 R_2 WHERE R_1.timestamp >11/3/2010 AND R_2.timestamp >11-3-2010 AND R_1.timestamp <12/1/2010 AND R_2.timestamp >12-1-2010 AND R_1.latitude>2 AND R_1.longitude>2 and R_2.longitude>3 AND R_2.longitude>3 and R1.OBJ_TYPE=1 AND R2.OBJ_TYPE=1*

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1 WHERE R1.time_stamp >to_date('08-dec-2010 09:41:30 PM','dd/mm/yyyyhh:mi:ss PM') AND R1.time_stamp < to_date('08-dec-2010 09:41:32 PM','dd/mm/yyyyhh:mi:ss PM') And R1.latitude>2 AND R1.longitude>3 And R1.obj_type=1	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2 WHERE R2.time_stamp >to_date('08-dec-2010 09:41:30 PM','dd/mm/yyyyhh:mi:ss PM') AND R2.time_stamp < to_date('08-dec-2010 09:41:32 PM','dd/mm/yyyyhh:mi:ss PM') And R2.latitude>2 AND R2.longitude>3 And R2.obj_type=1	R_2	2	Null	Null	R_{21}	2
UNION		R_{11}	2	R_{21}	2	U_1	2
Move	Null	U_1	2	Null	Null	U_1	3

3. Get all “Data” locations within 12 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

SELECT UAV1.latitude, UAV2.latitude, UAV1.longitude, UAV2. Longitude FROM UAV1 R_1, UAV2 R_2 WHERE R_1.latitude>12 AND R_2.latitude> 12 AND R_1.longitude>12 AND R_2.longitude>12 AND R1.OBJ_DESC=”Data” and R2.OBJ_DESC=”Data”

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1 WHERE R1.latitude>12 AND R1.longitude>12 And R1.OBJ_DESC=”Data”	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2 WHERE R2.latitude>12 AND R2.longitude>12 And R2.OBJ_DESC=”Data”	R_2	2	Null	Null	R_{21}	2
Union		R_{11}	2	R_{21}	2	U_1	2
Project	Latitude, Longitude	U_1	2	Null	Null	P_1	2
Move		P_1	2			P_1	3

4. Get all “Image” locations within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

SELECT UAV1.latitude, UAV2.latitude, UAV1.longitude, UAV2. Longitude FROM UAV1 R_1, UAV2 R_2 WHERE R_1.latitude>2 AND R_2.latitude> 2 AND R_1.longitude>2 AND R_2.longitude>2 AND R1.OBJ_DESC=”Image” and R2.OBJ_DESC=”Image”

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1 WHERE R1.latitude>2 AND R1.longitude>2 And R1.OBJ_DESC=”Image”	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2 WHERE R2.latitude>2 AND R2.longitude>2 And R2.OBJ_DESC=”Image”	R_2	2	Null	Null	R_{21}	2
Union		R_{11}	2	R_{21}	2	U_1	2
Project	Latitude, Longitude	U_1	2	Null	Null	P_1	2
Move		P_1	2			P_1	3

5. Get the latitude, longitude, and elevation for any object of type rocket launcher in the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

SELECT UAV1.latitude, UAV2.latitude, UAV1.longitude, UAV2. Longitude FROM UAV1 R_1, UAV2 R_2 WHERE R_1.latitude>R_2.latitude AND R_1.longitude>R_2.longitude

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2	R_2	2	Null	Null	R_{21}	2
Join	Latitude> Latitude and Longitude > Longitude	R_{11}	2	R_{21}	2	J_1	2
Project	Latitude, Longitude	J1	2	-	-	P_1	
Move	Null	P_1	2	Null	Null	P_1	3

6. Get the current ceiling and visibility in the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

SELECT UAV1.obj_desc, UAV2.obj_desc FROM UAV1 R_1, UAV2 R_2 WHERE R_1.latitude>R_2.latitude AND R_1.longitude>R_2.longitude

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2	R_2	2	Null	Null	R_{21}	2
Join	Latitude> Latitude and Longitude > Longitude	R_{11}	2	R_{21}	2	J_1	2
Project	OBJ_DESC	J1	2	-	-	P_1	
Move	Null	P_1	2	Null	Null	P_1	3

7. Get all forward blue force positions within 2 NM of the area bounded by latitude1, longitude1 and latitude2, longitude2.

Query:

SELECT UAV1.latitude, UAV2.latitude, UAV1.longitude, UAV2. Longitude FROM UAV1 R_1, UAV2 R_2 WHERE R_1.latitude>R_2.latitude AND R_1.longitude>R_2.longitude AND R1.OBJ_DESC="blue force" and R2.OBJ_DESC="blue force" AND R_1.latitude>2 AND R_2.latitude> 2 AND R_1.longitude>2 AND R_2.longitude>2

Query Plan:

Operation	Sub SQL& JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 R1 WHERE R1.latitude>2 AND R1.longitude>2 And R1.OBJ_DESC="Image"	R_1	1	Null	Null	R_{11}	1
Move	Null	R_{11}	1	Null	Null	R_{11}	2
Select	SELECT * FROM UAV2 R2 WHERE R2.latitude>2 AND R2.longitude>2 And R2.OBJ_DESC="Image"	R_2	2	Null	Null	R_{21}	2
UNION		R_{11}	2	R_{21}	2	U_1	2
Project	Latitude, Longitude	U1	2	-	-	P_1	
Move	Null	P_1	2	Null	Null	P_1	3

8. A sample query depicting three joins.

Query:

*"SELECT * FROM UAV1 R_1, UAV2 R_2, UAV3 R_3, UAV0 R_0 WHERE R_1.timestamp LIKE k AND R_2.timestamp LIKE j AND R_3.Obj_type='1' AND R_1.latitude>R_2.latitude AND R_2.longitude> R_3.longitude and R_0.latitude>R_3.latitude*

Query Plan:

Operation	Sub SQL & JOINS	Operand1	Operand1 Loc	Operand 2	Operand 2 Loc	Result Name	Result Loc
Select	SELECT * FROM UAV1 WHERE TIME_STAMP LIKE k	R1	1	Null	Null	R11	1
Move	Null	R11	1	Null	Null	R11	2
Select	SELECT * FROM UAV2 WHERE TIME_STAMP LIKE j	R2	2	Null	Null	R21	2
Join	Latitude > Latitude	R11	2	R21	2	J1	2
Move	Null	J1	2	Null	Null	J1	3
Select	SELECT * FROM UAV3 WHERE OBJ_TYPE=1	R3	3	Null	Null	R31	3
Join	Longitude > Longitude	J1	3	R31	3	J2	3
Move	Null	J2	3	Null	Null	J2	0
Select	SELECT * from UAVO WHERE OBJ_TYPE=1	R0	0	Null	Null	R01	0
Join	Latitude>Latitude	J2	0	R01	0	J3	0

8.6 SIMULATION RESULTS

Simulations are done considering a network of 12 UAVs in an area of 140 X 140 NM^2. Our performance metrics include 1) Average connectivity & Average Reach-ability for routing, 2) Average roundtrip time, Average hop count and Replica hit% for Replication and 3) Average Query execution time for analyzing the behavior of move and join operations in static query plans with respect to our architecture.

Simulation Environment: We have built a simulation environment in JAVA applet and have used Databases for UAV memory. The simulation parameters are listed as follows

Parameter	**Default Value**	**Range**
Simulation Area	140 X 140 NM^2	140X140 - 245X245NM^2
Database size n	50	10-50
Data size	1 MB	
Number of nodes	12	
Bandwidth	1Mbps	
Transmission Range	300 meters	
Velocity	.8 NM/second	.8 - 2.4 NM/second
Query generation time	1 second	

8.6.1 ROUTING

We did the following experiments to evaluate routing with respect to the current simulation system

a) Mobility VS Average Reach-ability.

b) % increase in simulation area VS Average Reach-ability

c) Time period of alive messages VS % Path Efficiency

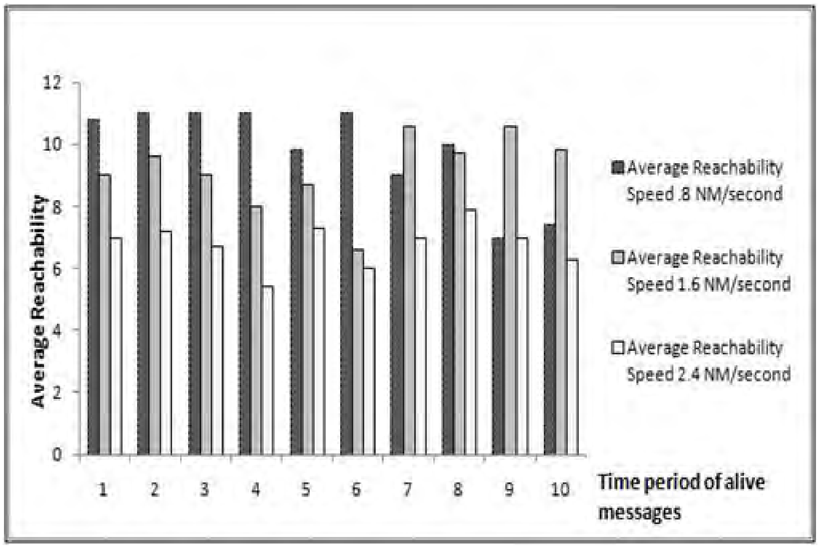
d) Mobility Vs Average Connectivity

a) Mobility VS Average Reach-ability:

The first experiment is to find the average number of UAV reachable in the network. Reach-ability is an important performance metric, which helps to analyze the success of UAV's contacting other UAV's for data. The availability of data in the network decreases proportionally with decrease in average reach-ability. We compute the average reach-ability as follows

Average Reach-ability = Sum of nodes reachable from all nodes / Total no of nodes.

The reach-ability changes with respect to change in network topology. In this experiment we calculate reach-ability at various time intervals with a different network topology to identify at most reach-ability for maximum number of UAVs (12) at various speeds. Every interval on x-axis constitutes average reach-ability computed every 20 seconds. Thus, this experiment is done for 100 different network topologies. On average, the reach-ability decreases with increase in speed, with maximum reach-ability of 9.8 is achieved at a speed of .8 nautical miles /second (equivalent to .92 miles/second).



The 95% confidence interval for each samples across the x axis are shown as follows

Speed\Interval	1	2	3	4	5
.8 NM	10.8±.4	11±0	11±0	11±0	9.8±.6
1.6 NM	9± 1.2	9.6 ±.6	9±1.2	8±1.8	8.7±2.1
2.4 NM	7±2	7.2±1.6	6.7±1.8	5.4±1.8	7.3±1.8
Speed\Interval	6	7	8	9	10
.8 NM	11±0	9±1.1	10±.7	7±0	7.4±.8
1.6 NM	6.6±2.1	10.6±.5	9.7±1.7	10.6±.5	9.8±1
2.4 NM	6±1.3	7±1.7	7.9±1.8	7±2.0	6.3±2.4

b) % increase in simulation area VS Average Reach-ability

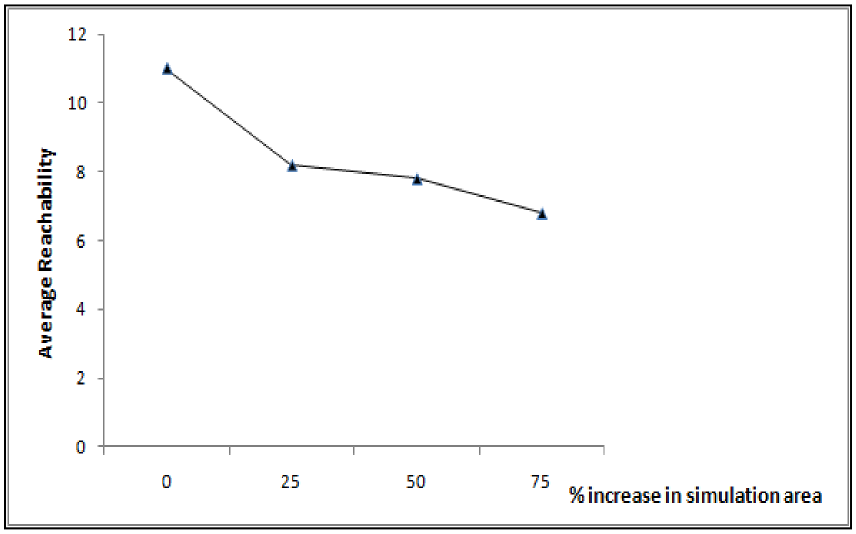
We evaluated Average Reach-ability with respect to simulation area. We increased the simulation area in terms of 25%, 50% and 75%.We have initial simulation area of 140X140 NM. We increased

a) 25% = 175X175 NM

b) 50% =210X210 NM

c) 75% = 245X245 NM

The Average reach-ability decreases with increase in area size. This is due to the wider placement of UAVs in larger areas, which causes too many disconnections forming isolated UAVs. The results are plotted and graph is shown as follows



The 95% confidence interval for samples across x axis are shown as follows

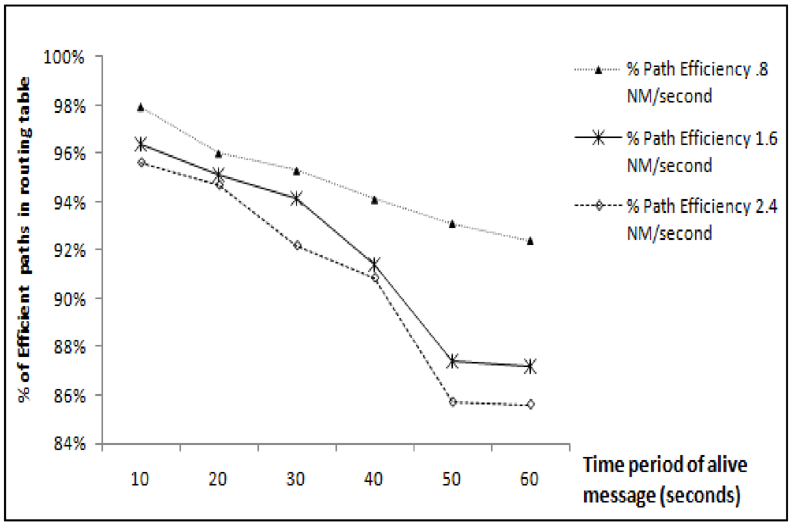
% increase in simulation area			
0	25%	50%	75%
11±.0	8.2±1.6	7.2±1.9	6.8±2.1

c) Time period of Alive messages vs % Path efficiency:

In this experiment, we try to find the best time-period to broadcast ALIVE messages. ALIVE messages contain the updated link information, which is used to reconstruct the routing table. The time interval in which the alive messages are sent should be optimal, such that the available path to send data is valid and also the shortest path to the destination. We evaluate the %path efficiency for different movement models sending alive messages at various time intervals between 10 and 60 seconds for 12 UAV's. We determine the path efficiency of the routing table based on the formula below:

% Path Efficiency= (no. of valid and shortest paths in routing table/ Total no. of paths) * 100

The topology change varies with respect to speed, hence we evaluate with respect to speed. The delay in ALIVE messages increases the delay in updates, which causes the construction of erroneous routing table; hence there is a decrease in efficiency with increase in time interval of alive messages.



At 10 & 20 seconds interval, updates are fast, hence the path in the routing table are almost valid bringing high efficiency irrespective of speed. At high mobility (1.6 & 2.2 miles/second), network topology changes rapidly and updates are also slow at 50 & 60 seconds. Hence, the path efficiency is very less as compared to path efficiency at .8 miles/second. We conclude that 30 seconds as ideal Alive message broadcast time since broadcast at reduced time interval cause too much congestion in network. Moreover, efficiency at 30 seconds is upright, i.e. the difference between path efficiencies at various speed is not that high.

The 95% confidence intervals of the plotted samples are shown as follows

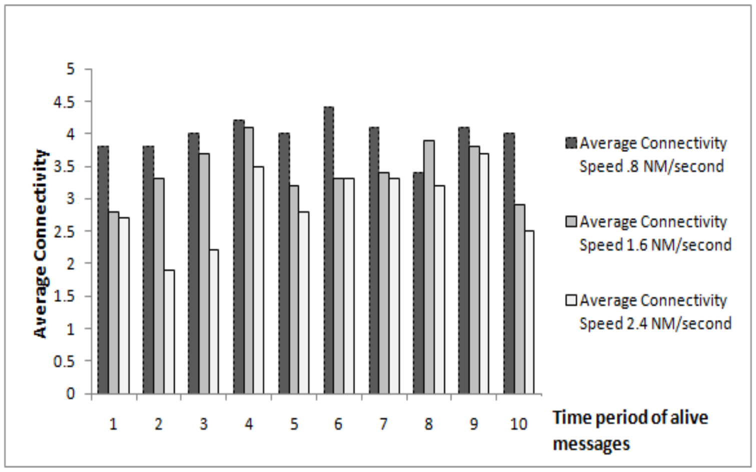
Time period of Alive messages						
	10	20	30	40	50	60
.8	97.9±.9	96±3.8	95.3±3.4	94±5.9	93±5.8	92.4±6
1.6	96.3± 2.4	95 ±3.1	94.1±3.6	91±4.4	87±6.4	87.2±3
2.4	95.6±2.1	94.8±2.1	92.2±5.3	91±4.7	86±5.9	85.6±5

d) Mobility VS Average Connectivity:

With respect to connections, we are restricting the connections for "UAV 0" to 3; the remaining UAV's can have as many connections depending on the numbers of UAV lying within their transmission range.

Average Connectivity=Total no. of connections for each node/Total no. of nodes
The connectivity changes with respect to change in network topology. In this experiment we calculate connectivity at various time intervals with a different network topology to identify at most connectivity

for maximum number of UAVs (12) at various speeds. Every interval on x-axis constitutes average connectivity computed every 20 seconds. Thus, this experiment is done for 100 different network topologies.



The average connectivity decreases with increase in speed, it is obvious that high mobility causes too many disconnections decreasing the connectivity. The connectivity at 0.8 NM/second mobility is maximum with an average of 3.98, which means for 12 UAVs every UAV has degree 4 as an average at .8NM/second.

The 95% confidence interval for the samples are shown as follows

Speed\Interval	1	2	3	4	5
.8 NM	3.8±.26	3.8±.26	4±0	4.2±.26	4±0
1.6 NM	2.8± .39	3.3 ±.41	3.7±.83	4.1±.20	3.2±.39
2.4 NM	2.7±.29	1.9±.20	2.2±.40	3.5±.33	2.8±.26
Speed\Interval	**6**	**7**	**8**	**9**	**10**
.8 NM	4.4±.32	4.1±.19	3.4±.32	4.1±.35	4±0
1.6 NM	3.3±.41	3.4±.32	3.9±.35	3.8±.39	2.9±.73
2.4 NM	3.3±.71	3.3±.41	3.2±.39	3.7±.65	2.5±.43

8.6.2 REPLICATION

We have two access models for validating the replication scheme.

***I)** Access based on Load balancing Score Function:(ABL)*

$$Score_N = [(Size_{Data} / BW_{min}) + hopcount-1] + \sum_{i=0}^{n} LWi * P_i$$

$Size_{Data} \leftarrow$ Size of the requested data item.

$BW_{min} \leftarrow$ Minimum bandwidth across the path between source to destination.

$LW_i \leftarrow$ Load Weight at node i, higher the load causes more time for query to get Processed.

$$P_i \leftarrow \begin{cases} 4 & \text{N is the destination (or) IS} \\ 1 & \text{N is Intermediate} \end{cases}$$

$IS \leftarrow$ refers to the intermediate data source, which lies at a shorter distance along the path to another data source for the same data.

With respect to above replication scheme, there will be two copies for any data item, request will be send to UAV having a lesser $Score_N$. We apply random load weights at each node; our load weight varies between 1 to 3. We have kept the delay due to single load as 10 milliseconds

***II)** Access based on reduced hop count: (ABR)*

Request will be send to UAV, which can be reached through lesser number of hops from source UAV.

We did the following set of experiments to validate replication scheme

a) Number of data items VS Average hop count.

b) Number of data items VS Average roundtrip time.

c) Number of data items VS Cache hit ratio.

a) Number of data items VS Average hop count

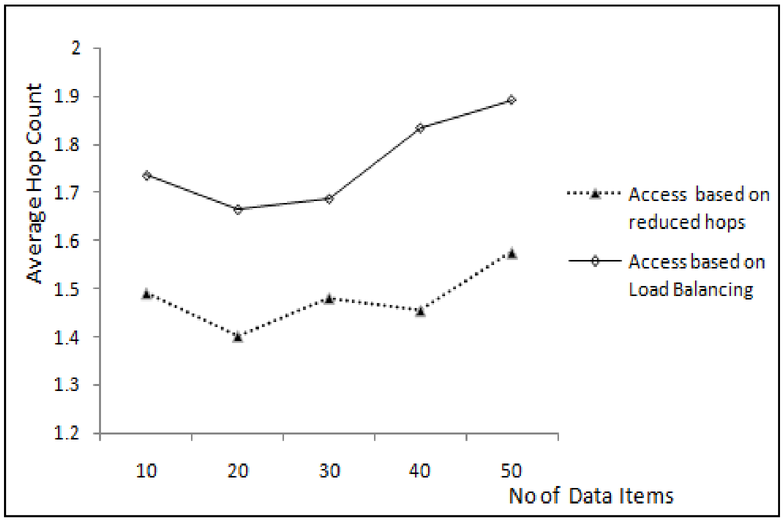
Average hop count is one of the ideal performance metrics, which helps to analyze the efficacy of data access due to replication.

Average hop count= $\sum_{i=1}^{Nq} \mathbf{Hi} / \mathbf{N_q}$

$N_q \leftarrow$ number of queries

$H_i \leftarrow$ hop count to reach the destination

We validate Average hop count with respect to the two access models (ABL & ABR). We measure the average hop count by varying the number of data items from 10 to 50. We follow ZIP-f distribution based query access with K=15 and α=0.8. The number of queries increases exponentially with increase in number of data items. We execute the average hop count formula for each case and the results are plotted as follows.



ABR favors hop count; hence the average hop count is less compared to ABL for all the cases. ABL favors response time. Though ABL has hop count in its equation, distant source might be chosen based on reduced load across the path and lesser estimated response time. ABR never cares about load or response time and considers only hop count. Hence, ABR provides better hop count as compared to ABL.

The 95 % confidence interval for samples in x axis is as follows

No of Data items					
	10	20	30	40	50
ABL	1.74±.18	1.7±.14	1.7±.25	1.83±.2	1.9±.18
ABR	1.49± .17	1.39 ±.09	1.48±.11	1.45±.09	1.57±.15

b) Number of data items VS Average Round trip time

Average Roundtrip time is also one another performance metric determining the efficacy of data access. This performance metric too, is highly dependent on the access models (ABL & ABR).

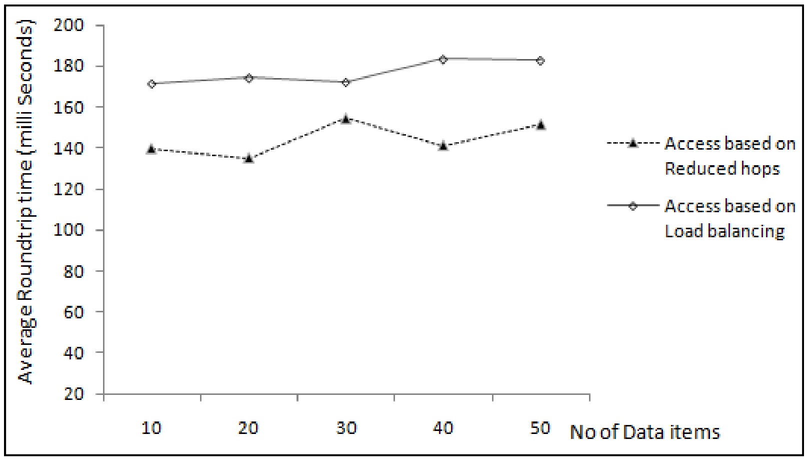
Average Round trip time= (1/N$_q$) * $\sum_{i=1}^{Nq}(Tr - Trs)$

N_q ← number of queries

T_r ←time stamp in which request is send.

T_{rs} ← time stamp in which the first initial response is received.

We measure the average round trip time by varying the number of data items from 10 to 50. We follow ZIP-f distribution based query access with K=15 and α=0.8. We execute the average roundtrip time formula for each case and the results are plotted. ABR has better roundtrip time since it chooses data from nodes at shorter distance; the first part of ABL score function chooses source from which data can be retrieved at a shorter response time. But in case of roundtrip time, we consider only the first initial response. Since ABL has increased hop count, the delay due to load across the path is high as compared to ABR. Since average hop count gives preference to shorter hops, the load is very less and the delay to get first initial response is less. Thus, the average roundtrip time of ABR is less as compared to ABL.



The 95% confidence interval for samples is as follows

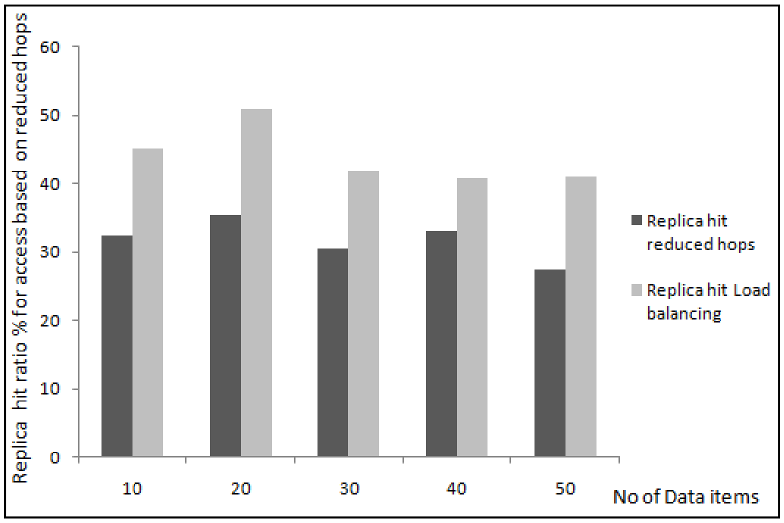
No of Data items					
	10	20	30	40	50
ABL	171.4±32.8	174.2±27	172.2±27	183.3±29.6	182.8±18.9
ABR	139.8± 23.1	135.2 ±8	154.4±13	141.3±15.4	151.6±24.1

c) Number of data items Vs Cache hit ratio:

In this experiment, we try to find effectiveness of the replication algorithm. We analyzed the replica hit %, by varying the number of data items from 10 to 50. The formula is given below:

Replica Data hit % = No of replica hits/ N_q.

$N_q \leftarrow$ number of queries



ABR give preferences to shorter hops irrespective of replicas, moreover ABL uses bandwidth, which is used by the replication algorithm too. Hence the replica hit % of ABL is good compared to ABR.
The 95% confidence interval for samples are shown as follows

	No of Data items				
	10	20	30	40	50
ABL	45.17±3.34	50.9±4.9	41.8±4.15	40.9±4.52	40.98±6.24
ABR	32.46±15.6	35.4 ±8.1	30.4±7.6	33.1±6.9	27.55±4.67

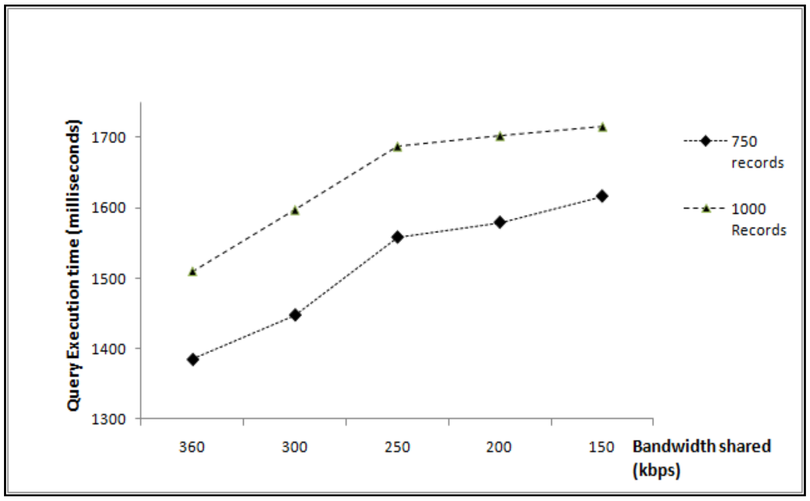
8.6.3 *STATIC QUERY EXECUTION*

We did the following set of experiments for evaluating replication with respect to the current simulation system.

a) Query Execution time Vs Bandwidth shared & No of data items
b) Query Execution time Vs Number of data items for various queries
c) Number of data items Vs Number of joins

a) Number of data items Vs Query Execution time:

We evaluated the query execution time with 3 UAVs for query plan shown in table 1.3 by increasing the number of data items and bandwidth shared by them from the broker. The query execution time increases with increase in number of data items, which is due to the join operation; hence the query execution time for 750 records is less than 1000 records. Similarly, the query execution time increases with reduction in bandwidth shared, this is due to the delay incurred by the move operation on transferring the query results. The plotted results are as follows

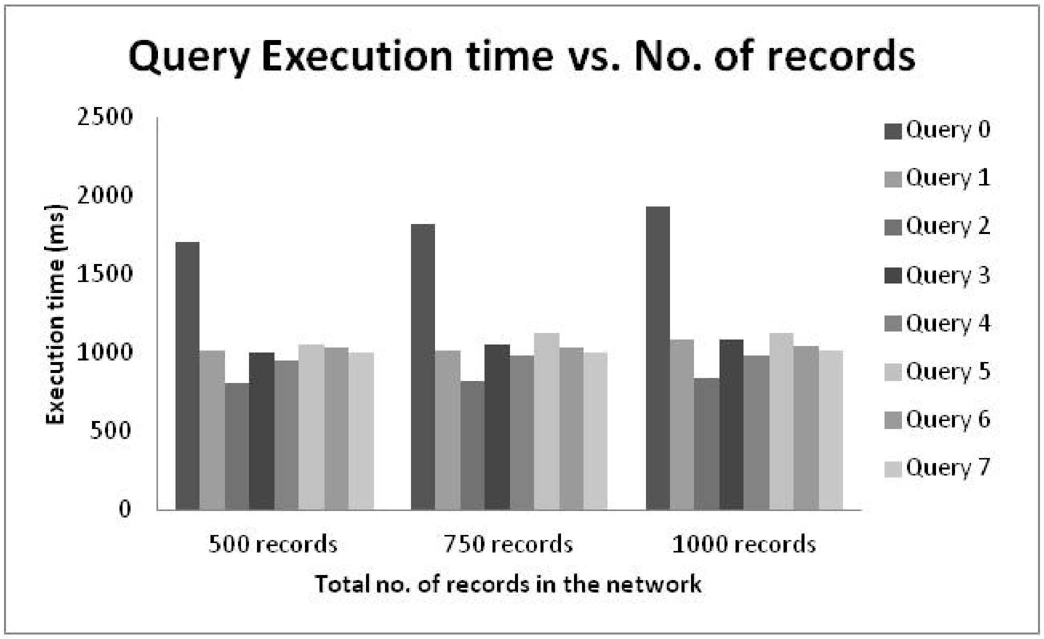


The 95% confidence intervals for samples are as follows

Bandwidth shared			
	360	300	250
750	1384.8±72.8	1448.4±129.4	1558.4±49.2
1000	1509.8±102.7	1597±23.1	1686.6±46.6
	200	150	
750	1579.4±147.5	1616.8±73.5	
1000	1702.2±92.3	1715.4±59.1	

b) Query Execution time Vs Number of data items for various queries

We evaluated the query execution time for all the query plans in section (**E)** by varying the number of data items. At each step, we increase the number of records in the network by capturing more data. The number of records in the network is increased from 500 to 1000 records and various queries are executed. The query execution time is found and the results of the above experiment are plotted as shown below



The query execution time increases as the number of records in the network increases. This is because the time taken to join tables, combine tables and transfer data between UAV's increases with the increase in total number of records. We compare the execution times for various benchmark queries and compare them in the above graph. The increase due to number of records is not large because only the meta-data is transferred between UAV's.

c) Number of data items Vs Number of joins

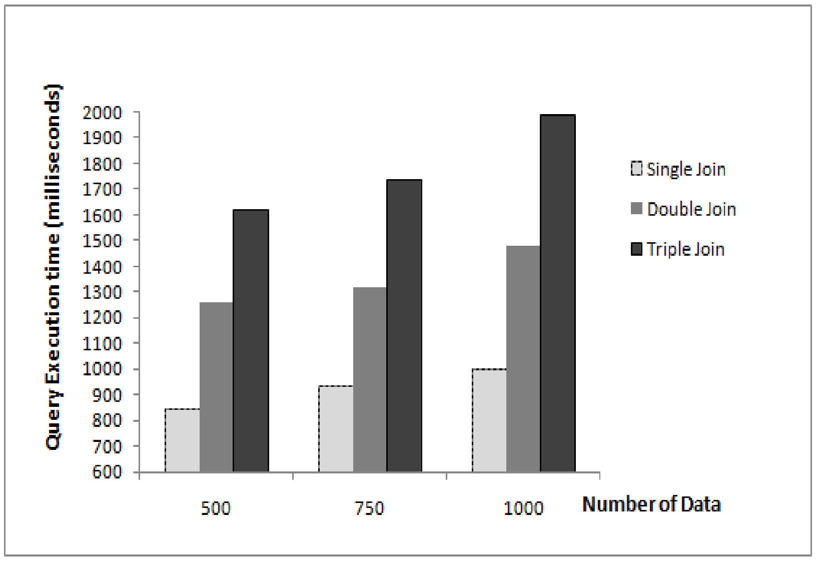
We evaluated the query execution time for various numbers of joins in the query; the following query plans in Section **E** have the corresponding number of joins

Query 5← Single Join

Query 0←Double Join

Query 8←Triple Join

We vary the number of records for each set and the results are plotted as follows



The query execution time increases will increase with respect to increase in number of joins in the query plan. Moreover, the increase in data item too increases the query execution time, due to more processing time while joining the data items.

The 95% confidence interval for the above plotted results is as follows

Bandwidth shared			
	Single	Double	Triple
500	842.4±98.14	1263±73.3	1616.6±91.5
750	935.6±51.9	1321.2±103.7	1735.2±70.83
1000	999.6±40.3	1479.4±72.4	1988.2±65.17

9. SOFTWARE DEVELOPMENT

Two pieces of software have been developed for this project:

A query plan generator described in Section 3. This includes the software for generating connectivity configurations. Experimental results are in Section 3.6.

A simulator described in Section 5. Experimental results of the simulator are also described in Section 5.

All the software developed for this project will be delivered to AFRL on a DVD or CD Rom. They can also be downloaded by right-clicking and downloading on the following (hidden) links:

http://itlab.uta.edu/downloads/QPGSETUP.zip
http://itlab.uta.edu/downloads/SIMULATORSETUP.zip
http://itlab.uta.edu/downloads/setup-instructions(qpg).zip
http://itlab.uta.edu/downloads/setup-instructions(simulator).zip

10. RESULTS AND DISCUSSIONS

Collaborative airborne, ground/surface based, manned and unmanned missions are expected to become more prevalent in the future.

We have provided an insight into some of the fundamental issues with regard to scalable, fault tolerant repositories. We have provided a use case analysis that allows the focusing of technological needs in a complex tactical scenario.

We have provided insight into what the Metadata management should look like, optimized Query plan generation algorithms and heuristics to this scenario which differs from traditional distributed query processing, and approaches to Replication management. We have provided simulations of the above that allow easier visualization and understanding of some of these complex issues. The software developed has also been provided.

This research has enabled an understanding of the issues, and has raised a number of other fruitful research needs:

Collaboration of teams of manned and unmanned systems will require a different type of command and control system. This next generation C2 system must work at a number of different levels. It will need to provide planning, strategic control, and tactical control. It will need to understand that systems and platforms may embark on missions, without access to communications, and then rejoin.

The data repositories will be distributed, highly mobile, and dynamic. The ISR requirements dictate that massive amounts of data will need to be collected and analyzed autonomously in tactical timeframes

In summary, this project has established the foundation for this problem by identifying the sub problems, issues in each sub problem, and has investigated how they all come together towards the solution of the initial problem. This effort is by no means complete. In each section, we have articulated the need for extending proposed solutions to reach a practically useful approach/solution.

Furthermore, we have not addressed all the problems identified in the proposed architecture shown in Figure 1. There are a number of important problems that need to be addressed to obtain a complete solution. We will be happy to explore opportunities with AFRL or other agencies to continue this work.

In this project, we have implemented proof-principle systems to show the effectiveness of the proposed approaches. The simulator has been extended to incorporate features specific to this scenario and analysis has been performed. The next logical step is to create a prototype by bringing all the components together and then move towards a testbed. The team has been chosen from the outset keeping the long term goals in mind and team is well-positioned for additional work on this project.

11. CONCLUSIONS

The project involved the development of a middleware for fault-tolerant computing in a distributed network of UAVs, each carrying data repositories. Each node or UAV in the network is host to metadata, sourced at the node as well as replicated. The developed middleware serves the main purpose of responding to queries. We have developed methods for handling use case scenarios, metadata management, replicated data management, query plan generation algorithms, and network management.

Currently, a static plan is generated for each query and the query is processed sequentially by performing a sequence of operations at different nodes. This can be further improved by generating a dynamic plan at each node where the query is processed to accommodate intermittent connectivity among nodes. Furthermore, a query plan can be executed in parallel (using either static or dynamic plans) to improve response time. This can be extended to multi-copy and partial replica again to accommodate intermittent connectivity. Above-mentioned extensions will add complexity to both Meta data management and query processing.

In conclusion, there are a number of important problems that need to be addressed to obtain a complete solution. The team will be happy to explore opportunities with AFRL or other agencies to continue this work.

12. REFERENCES

1. Air Land Sea Bulletin, *Close Air Support*, Issue 2010-2, May 2010.
2. Bauer, Major Zdenek (Czech Air Force), *Combat Controllers and UAVs Together*, USAF Air University, April 2007.
3. Defense Science Board Task Force, *Integrated Fire Support in the Battlespace*, Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, October 2004.
4. Drury, Jill L. and Darling, Erika, *A "Thin-Slicing" Approach to Understanding Cognitive Challenges in Complex Command and Control*, The MITRE Corporation, 3 January 2007.
5. Gruetzmacher, Commander Jeffrey K. (USN), Holtery, Lieutenant Colonel Michelle Joerin (USA), Putney, Major Jonathan R. (USAF), *Fratricide: The Ultimate Cost of Joint interoperability Failure*, Joint Forces Staff College, 11 June 2002.
6. Hume, Colonel David B. (USAF), *Command, Control and Integration of Weaponized Unmanned Aircraft into the Air-to-Ground System*, USAF Air War College, 23 February 2007.
7. Joint Chiefs of Staff, *Doctrine for Joint Fire Support*, Joint Publication 3-09, Washington, DC, 30 June 2010.
8. Joint Chiefs of Staff, *Joint Tactics, Techniques, and Procedures for Close Air Support (CAS*), Joint Publication 3-09.3, Washington, DC, 8 July 2009.
9. Marine Corps, *Close Air Support*, Marine Corps Warfighting Publication (MCWP) 3-23.1, July 1998.
10. Martin, Captain D. E., *The Future of Marine Corps Close Air Support: The Urban Environment*, USMC Command and Staff College, 7 February 2006.
11. Pirnie B., Vick A., Grissom A., Mueller K., and Orletsky, D., *Beyond Close Air Support: Forging a new air-ground partnership*, RAND Corporation, 25 April 2005.
12. Simpson B., Rouff C., Roberts J., Edwards G., *An Autonomic system for Close Air Support*, 2009 IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems.
13. Taliaferro, Major Jeffrey B., *Coordinate-guided Weapons in Close Air Support: An Evaluation of Risk*, USAF Air University, June 2003.
14. Unterreiner, Commander Ronald J. (USN), Brelsford, Major Jeffrey A., Findlay, Major Richard J. (USMC), Hunnell, Major John F., Wagner, Major Michael F., *Close Air Support (CAS) in 2025*, USAF Air University, August 1996.
15. US Army, *Attack Helicopter Operations*, Field Manual 1-112, 2 April 1997.
16. US Army, *Army Unmanned Aircraft, System Operations*, FMI 3-04.155, April 2006.

The following paper has been published by the team of investigators

1. Hemanth Meka, Lekshmi Manian Chidambaram, Sanjay Madria, Mohan Kumar, Mark Linderman and Sharma Chakravarthy**, ROMAN: Routing and Opportunistic Management of Airborne Networks,** , to appear in IEEE proceedings of International Symposium on Collaborative Technologies and Systems (CTS 2011), Philadelphia, USA.
2. **M. Kumar**, S. Chakravarthy, S. Madria, M. Linderman and W. Naqvi, Middleware for Supporting Content Sharing in Dynamic Networks, The 2011 Military Communications Conference (Unclassified Papers), Baltimore, November 2011.

List of Acronyms

AAA – Anti Aircraft Artillery

ACK – Acknowledgement

AWAC – Airborne Warning and Control System

BER – Bit Error Rate

CNF - Conjunctive Normal Form

DMZ - De Militarized Zone

FLOT - Forward Line Own Troops
ID - Identifier

JAOC - Joint Air Operations Center

JTAC – Joint Terminal Attack Controller

LS – Link Stability

MIO - Metadata Information Object

NAK – Negative Acknowledgement

NM – Nautical Mile

RSS – Received Signal Strength

SAM - Surface to Air Missile

SEAD - Suppression of Enemy Air Defenses (SEAD)

SOA - Service Oriented Architecture

SPJ – Select-Project-Join

SQL – Structured Query Language

TCP – Transport Control protocol

UAV - Unmanned Aerial Vehicles

UDP – User Datagram protocol